CRANFIELD UNIVERSITY

J HARRINGTON

AN INTELLIGENT NEGOTIATION BASED FRAMEWORK TO SUPPORT CONCURRENT ENGINEERING PRINCIPLES IN THE ENGINEERING DESIGN OF PROCESS PLANT

SCHOOL OF INDUSTRIAL AND MANUFACTURING SCIENCE

PhD THESIS

CRANFIELD UNIVERSITY

SCHOOL OF INDUSTRIAL AND MANUFACTURING SCIENCE

PhD THESIS

Academic Years 1992-6

J HARRINGTON

An Intelligent Negotiation Based Framework to Support Concurrent Engineering Principles in the Engineering Design of Process Plant

Supervisor: H Soltan

October 1996

This thesis is submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

Abstract

The traditional approach to the engineering design of process plant is highly sequential with decisions made early in the design phase having a large knock on effect to downstream design processes. A lack of consideration to downstream concerns will either result in design re-work or compromise. Concurrent engineering has been proposed as a design method for resolving the problems inherent in the sequential design process by bringing the different engineering disciplines together at key decision points in the design process, thereby preventing design problems before they occur.

Computational support for concurrent engineering aims to develop tools to help team members in sharing knowledge and keep track of the others' needs, constraints, decisions and assumptions [Cutkosky, et.al. 93]. Such systems would enable engineering disciplines from each of the design life-cycle stages to communicate and review design strategy. As a group they would be able to explore design alternatives in search of a good solution [Talukdar, Fenves 89].

Knowledge based systems can support the engineering design process by providing advice that accounts for the global concerns. It is argued that such a system should be distributed, due to the problems in maintaining a single large knowledge base, and computational power required to operate a single system. However, wherever expertise is distributed, conflict exists that has to be resolved.

The aims of this research are to identify the needs of a computational support environment to aid concurrent engineering design, and to develop a framework to enable disparate design systems to cooperate and produce designs acceptable from the global viewpoint. The 'needs' were identified from a study of the engineering design process, and a detailed analysis into the design and selection of pumping systems to provide a rich example of the problems faced in a specific design process.

Cooperation is achieved through 'Negotiation', which resolves conflicts between the various objectives involved in design and is a central theme of this research. Through the provision of a framework to support negotiation the aim is to provide the basis on which individual design programs can cooperate to produce rational designs from a global perspective, thereby bringing life cycle design advice to the earlier design stages.

Acknowledgements

I would like to thank the following organisations and individuals for providing me with their time in supporting this research effort:

SFK Technology Ltd
The British Hydromechanics Research Group
Hossein Soltan, my Cranfield Supervisor
Mark Forskitt, Internal SFK Supervisor
Geoff Conroy, Project Management Consultant
Derek Burgoyne, Pump Consultant and my Industrial Supervisor
Roger Mallinson & Jeremy Illidge, ICI Engineering
ICI
Brown & Root Braun
Foster Wheeler Engineering
Stone & Webster
HMD Pumps Ltd
Rolls Royce and Associates
Ingersoll Rand

I would also like to thank SERC, and the DTI for sponsoring this research.

In addition to those who provided an input to this research, I would like to thank those people that helped and put up with me through some difficult times. I would therefore like to thank..

My family for their prayers.., Lisa for everything beautiful.., Sandra and Walt for having Lisa.., my Aunt Sally for helping me with the binding, Hossein - my supervisor - for putting up with me for such a long period of time.., Andy G. for picking me up.., Barclays for putting me down.., David...a bloke.., Rachel for her high spirits.., Carl Myhill for going through it with me.., Lynne Badr for looking out for me.., Mike for his curries.., Rob for his house.., Dianne for her politics.., Caroline for letting me have the time off.., Derek Burgoyne for his wisdom.., Mark Forskitt for being Mark.., Welly for his alternative point of view.., Ruth for her enthusiasm.., Chris for his optimism.., Mike B for his pessimism.., Andy W for his "I'm not sure whether to be optimistic" style of optimism.., Simon for his interest in interest.., Steve Field for the loan of his comics.., Steve C for making me look good driving a Polo.., Martin for his Christmas cheer.., Paula for her alcoholic exploits.., Bill for being there.., Karen for not being there when I'm there.., Tony for putting up with me lying on my resource sheets.., Sainsburys for their bonnoffee pie flavoured ice cream.., and everyone who has had to put up with me or met me in the last four years...

Table of Contents

Appendices

List of figures

Chapter I. Introduction

I.I. Overview

The aim of this chapter is to introduce the research, the rationale behind the research - (the problem to be solved) - and the steps taken to develop and test an approach to solve the problem.

The engineering design of process plants is a complex activity requiring the skills of a wide range of experts. This chapter shows how concurrent engineering as a principle can aid the improvement of such design.

Concurrent engineering does have inherent problems which are outlined, as are the technological difficulties of developing systems which utilize the concurrent engineering design philosophy.

Computer systems have aided design in many ways. Through the integration of design software, engineers have been able to share information quickly regarding the design, therefore identifying problems earlier in the design phase. This chapter briefly introduces the concept that for intelligent KB systems to progress in design support, the resolution and management of conflict between these systems has to be addressed, this is referred to as 'Negotiation'. Negotiation is discussed in relation to the idea of interfacing software systems. It is based on the premise that to cooperate fully, a mechanism must be in place to resolve differences, rather than just enable the different systems to coordinate (follow a protocol) and understand the same language (data representation) - although this in itself is a difficult problem.

I.2. The industrial problem

The traditional approach to the engineering design of process plants is highly sequential, with design tasks being performed without due consideration of the operation, maintainability and other areas in plant design. The general problems faced with this sequential approach to design and manufacture have been well documented [Tomarchio 91]. Products take too long to develop, cost too much to produce, and often do not perform as promised or expected [Cleetus 92]. The major cause of such failings is the lack of integration between disparate design disciplines.They neither understand nor consider the goals and constraints of other disciplines involved in design. These misunderstandings between disciplines cause significant amounts of design re-work, and design compromise in order to complete on time (Figure 2).

The more experienced engineers - sometimes referred to as the 'white tops' - who typically

have experience in a number of design departments - produce better designs because they can bring their broad experience to bear on the design issue. This massive pool of experience enables them to identify and assess the impact of their design decisions on the rest of the design. Consequently, designs more readily satisfy the demands of other design disciplines, are produced in a shorter time period-due to less iteration and design re-work, and the result is of better quality due to less design compromise. The plant is therefore cheaper to design and of improved quality with regard to performance, safety, and ease of operation. Fifteen years ago Conaco used to employ fifty engineers in its design department. Now they only employ five highly experienced engineers in the department yet are more productive. They have the experience and skills to cover the whole design process. Additionally there is considerably less communication than in a larger team therefore improved consistency, and they farm more of the routine design effort out to the vendors. BP has also recognised the benefits of this approach and are adopting the same strategy.

The way in which the market in the engineering of plant is developing will put a greater strain on engineering design teams. There is a strong pull to reduce the time taken to design and develop a plant as well as reduce the overall cost. This means utilising as few engineers as possible in the most productive manner. The time constraint will impose more urgency in ensuring that the results from various stages of design are correct first time and therefore promoting design compromise in order to meet these deadlines. In recent years the economic climate has seen a large proportion of experienced, well qualified engineers leave the industry. This experience is not being maintained or captured within the engineering organisation in which they work. Costs have promoted the use of younger, therefore less experienced engineers on projects, which is likely to have an impact on the efficiency of design teams, and ultimately plant quality and safety.

1.3. Problems in the design of pumping systems

As a general statement, pump maintenance can account for up to 20% of the total maintenance cost of a chemical plant, depending on the corrosive nature of the plant, cost of materials of construction, and many other factors [Dormer, McKenna 73]. Of the pumps employed in the chemical industry, 80% of the pumps employed are withdrawn from service because of mechanical seal failures, with the remaining 20% caused by bearings, couplings, and other associated items [Barnard 92].

Pump reliability problems in the vast majority of cases occur due to two main factors. Firstly, around 90% of pumps used on peripheral services are over specified in terms of pressure. Secondly, many pumps are operating at duties well below their recommended minimum operating range [SfK 92c]. In many cases, the process of finding the best pump for the duty relied on incremental refits of the system. In other words, the maintenance costs for a pump would eventually rise to such an extent that it became more economical to simply replace it. The new pump would be specified differently in an attempt to

overcome the problems, and would improve the performance of the old pump [SfK 92c].

With operating experience, deficiencies in the original specification are uncovered along with deficiencies in the design of the equipment. Problems arise out of ignorance on behalf of the customer as to what he really wants, and what information is important to the vendor for his design. A vendor will supply a piece of equipment for what he thinks is necessary to do the job, often with ignorance of the application and service conditions, and often failing to ask for further details of the application [Barnard 92].

The problems inherent in pump selection would appear to benefit from improved communication and sharing of goals between the process engineer, mechanical engineer and pump vendor. The individual disciplines tend to design from their own viewpoint which leads to ideal solutions that often cannot be met exactly. This is noticeable between the process, mechanical and piping engineers and compromise solutions have to be made [SfK 92b]. It is thought that the best way of tackling problems with pump installations would be to ensure that the process engineers produce good process data sheets and general arrangement diagrams.

The vendors identify that many of the pump problems are due to the fact that they do not have access to the right level of detail [SfK 92b]. The mechanical engineer uses a précis of the process engineers work to formulate a specification sheet. Much of the process design intent is lost in this translation. It was noted that they would be quite happy to see the mechanical engineers bypassed completely. Additionally, many problems are derived from the fact that the process engineer does not always take pump operating conditions into account when he is designing the process. By the time the mechanical engineer or vendor spots the potential problems the design is already cast in stone.

Adoption of a knowledge based strategy to resolve these issues would also improve the support of design guidelines and reduce the loss of expertise within a company. The ICI design guides are complicated and require the engineer to have considerable expertise with pumps in order to use them. The contractors do not follow the design guides and they are too complex for most people to use. Engineers either do not read them, or do not understand the implications of what they read [SfK 92b]. The average age of design engineers in ICI is reducing and they are losing their most experienced personnel. Many problems exhibited in the past are reappearing as designers cut down on margins and as expertise is lost from the industry [SfK 92c]. Due to these problems, pump reliability is declining rather than improving, and many of the mistakes are elementary text book errors.

I.4. Concurrent Engineering

The goal in the process industry is to build plants with optimised capital cost, efficient to operate, and safe. This not only requires that the stages of design be highly and effectively integrated, but that the tasks run concurrently to as high a degree as possible [Reklaitis,

Preston 89].

Concurrent engineering is a method of achieving this goal through the use of multi-disciplined teams and have proven successes [Knodle 91][Dutcher 91][Tomarchio 91] in industries such as Aerospace and Automobile. Concurrent engineering addresses the entire way in which products are developed. It is a theory that aims to achieve the optimum design by ensuring that all engineers understand the limitations involved in the design and manufacturing process and account for these limitations in their work [Reklaitis, Preston 89].

A leading principle of concurrent engineering is to have proactive, 'end item' responsible, product development teams [Knodle 91]. Having a multi-disciplined design team will enable a design to be critiqued from the multiple task perspectives - similar to the capabilities of the *white top* engineers. The team will reduce the design re-work as problems can be identified earlier in the design stage and necessary corrections made. The design should progress through all the design stages without major complications. Identifying design problems at an earlier stage can have considerable cost benefits [Tomarchio 91] and reduce the compulsion to compromise a design.

Parallel working by all life cycle perspectives early in the design process is an important facet of concurrent engineering. Engineers must collaborate effectively in order to understand each others' requirements and ensure the optimum design with respect to all design viewpoints (Figure 5). Parallel working in the early stages of a project is more important due to the larger grained decisions that have to be made. Early decisions set the lower bounds on cost, time complexity and risk, as well as establishing the upper bounds on achievable product reliability and customer satisfaction [Cleetus 92]. Studies in an Aerospace industry have shown that by the time 5% of the product development time and cost have been expended, 85% of the product life cycle cost had been committed [Tomarchio 91].

1.5. Concurrent Engineering Problems

In order to implement concurrent engineering one needs to break down the procedural and physical brick walls that exist between the various disciplines. Problems concerning this task are similar in most industries. Concurrent engineering is a transformational change rather than incremental and can lead to feelings of insecurity [Tranfield, Smith 90]. Commitment of the management and workforce is imperative to the success of a project.

There is concern in the chemical industry [SfK 92a] that multi-disciplined design teams will lead to engineers with more generalised skills at a time when specialist expertise in particular areas is reducing. ICI have been asking the question *'how do we grow our new experts ?'*.

The partitioning and overlapping of design tasks has the inevitable penalty of inconsistency [Cleetus 92]. The inter-dependencies, problem dimensions, issues, and different criteria of the individuals are the inevitable cause of conflict which has to be resolved. Additionally, the human ability to adequately comprehend and evaluate design alternatives diminishes with the increasing complexity of inter-dependencies between the parallel design tasks [Cutkosky, et.al. 94][Anson, Jelassi 89]. Common human behavior to overcome these complex inter-dependencies is to ignore them, thereby serialising the design process and ignoring the impact of their design decisions on later design stages. A sub-conscious assessment of relevant design information - or engineer 'gut feeling' - has been identified as unacceptable given the increasingly complex technology and the increasing consequences of wrong decisions [Simmons 93]. It is clear that there is scope for improving design decisions by providing the decision maker with advice on the probable consequences of his decisions.

I.6. Computational Support for Concurrent Engineering

Computational support for concurrent engineering aims to develop tools to help team members in sharing knowledge and keep track of others' needs, constraints, decisions and assumptions [Cutkosky, et.al. 93]. Such systems would enable engineering disciplines from each of the design life-cycle stages to communicate and review design strategy. As a group they would be able to explore design alternatives in search of a good solution [Talukdar, Fenves 89].

In recent years there has been the development of a formidable array of engineering design tools such as CAD, Finite Element Analysis Tools, Simulators, and knowledge based design tools to help engineering designers. A flaw in the development of these tools is that they have been developed in the confines of a particular engineering discipline without regard to the needs of other engineering disciplines. To solve this problem there has been considerable investment in the integration of design tools in organisations through the development of common interfaces and data exchange standards. The integration of these packages will enable design data to be quickly transferred and reviewed by other engineering disciplines therefore reducing the time to identify and correct design faults. The integration of knowledge based systems to support the concurrent engineering philosophy however involves additional requirements. Besides the common knowledge representations that need to be employed, the provision of engineering advice requires proper consideration of the life cycle perspectives.

The traditional knowledge based approach of developing a single, consistent knowledge base to provide the life cycle perspective poses significant problems. Although designing a knowledge based system with a single knowledge base is certainly easier than integrating multiple knowledge bases [Steier, et.al. 93], the problems posed for knowledge elicitation, maintenance, and the addition of new knowledge are formidable. It has been stated that "it may be much easier to coordinate the activities of 20 medium sized machines than to

build a single machine that is 20 times as large" [Uma, et.al. 93].

In order to avoid the problems with traditional knowledge based systems, an approach to partitioning KBS systems into loosely coupled systems is required [Uma, et.al. 93]. Individual knowledge based systems operating in such a distributed environment have been termed *Agents*. Having single autonomous agents however poses the need for coordinating them towards solving complex design problems. Reasonable models of distributed expertise will dictate that agents hold incomplete local views of the problem [Davis, Smith 88]. Agents may posses knowledge unknown to other agents, maintain different beliefs and evaluation criteria, have incompatible knowledge representations, or may just be logically inconsistent with each other [Bond, Gasser 88]. Conflict is therefore inherent where expertise is distributed.

There are a variety of conflict resolution strategies that can be applied to resolve a conflict situation, both in the human workgroup and between disparate knowledge based agents. The strategies each have individual strengths and are appropriate in different situations. Negotiation has been proposed as a conflict resolution strategy in the sense that the roots of conflict are examined and rectified during negotiation [Pruitt 81]. In artificial intelligence terms, negotiation has been viewed as a conflict resolution and information exchange scheme [Bond, Gasser 88]. Negotiation can be viewed as a high level conflict resolution protocol, in that it identifies the key problem, maps the problem domain, and iteratively applies appropriate conflict resolution strategies until the conflict is resolved. In this sense the negotiation mechanism plays a coordinating role, rather than sitting alongside techniques such as compromise, constraint resolution, assumption surfacing and other conflict resolution techniques.

I.7. Computational Prototype

Knowledge based systems can support the engineering design process through the provision of expert design advice to the disciplines that make decisions that effect other downstream design tasks. A knowledge based system to provide engineering advice should ideally account for the global concerns when providing advice. Through the provision of advice in which downstream concerns have been accounted for, the engineer will have the opportunity to address the life cycle design issues, therefore realising a key benefit attributed to the concurrent engineering approach to design.

This research aims to identify and define a strategy for negotiation that will enable disparate knowledge based systems to cooperate. Through the provision of a framework to support negotiation the aim is to provide the basis on which tools to support concurrent engineering design can be built.

The result of this research is CDEX - the Concurrent Design EXpert. The basic requirements for the CDEX framework were derived from an analysis of the problems in

the process industry, the motivating factors, and the constraints imposed on the engineers. A detailed study was performed into the design and selection of pumping systems to provide a rich example of the problems faced in a particular design process and to aid in the support for computational knowledge based strategies to support engineering design. An important factor in the framework was for the agents to provide a reason why a particular solution was appropriate, rather than just accepting or rejecting a proposal, or by providing a single value which indicates the degree to which an option is supported. This understanding of the 'why' as well as the 'what' enables the framework to develop a rational solution to a design problem by attempting to account for all the interests involved.

The test of the CDEX design approach and the approach to resolving design conflict (termed 'negotiation') was performed using a real life engineering design meeting between two engineers. Both engineers had different functions with the engineering design organisation and had different priorities. The knowledge applied during the meeting by the engineers was classified by engineer, and two distinct knowledge bases developed. These knowledge bases conformed to a limited set of requirements needed for conflict resolution to take place. The framework enabled the disparate knowledge base systems to progress the design in a 'rational' fashion with regard to the options covered in the design meeting. The results were promising and the approach shows potential for the development of cooperating knowledge based systems in the field of engineering design.

1.8. Contribution to knowledge

The primary aim of this research was to develop a way in which knowledge based systems could support concurrent engineering principles in the design of process plants. In order to achieve this objective, the knowledge based systems required to be distributed, and the problem regarding the resolution of conflict - 'negotiation' - had to be resolved. The contribution to knowledge presented in this research has therefore mainly been in the areas presented for automated negotiation.

The following is a list of what one believes to be a contribution to knowledge:

i. The implementation of a set of human negotiation strategies in the computational domain.

   The approaches to resolving conflict applied in the human workgroups have been modelled using a prescriptive approach based on the development of an engineers' objective hierarchy, and tradeoffs in utility theory. The mapping between the techniques and computational utility theory is shown to be effective. The principle behind adopting the rather 'fuzzy' approach of human conflict resolution approaches (e.g. drop least important goals) is that the results of conflict resolution are more likely to be readily acceptable to the engineers.

ii. The development of a knowledge based strategy for negotiation that has the attributes required of a mechanism to support concurrent engineering design:

a/ copes with failure

b/ reviews the problem domain to determine the most appropriate strategy

c/ utilises the engineers' 'values' in reasoning about appropriate resolution strategies (an exchange of explicit knowledge is not required).

d/ applies strategies that can cope with conflicts among any number of agents.

e/ the agents' reasoning process can utilise individual technologies from mathematical engines and CAD systems to knowledge based systems. The framework does not require the agents themselves to account for rules and knowledge applied by other agents.

iii. The research has identified and classified the root cause of problems in a small part of the engineering design domain (the design and selection of pumping systems). This research has shown that the majority of problems can be prevented by better accounting of early design decisions on the later design stages (i.e. the application of concurrent engineering principles).

iv. A workable and extensible design framework has been developed to support the negotiation mechanism proposed. This framework is amenable to distribution as the approach applied in the framework does not dictate that the search process execute on a single processor.

An important factor in the framework is that the engineering agents can provide a reason why a particular solution is appropriate (in relation to the goals of design), rather than just accepting or rejecting a proposal, or by providing a single value which indicates the degree to which an option is supported. This understanding of the 'why' as well as the 'what' enables the framework to develop a rational solution to a design problem by attempting to account for all the interests involved. The framework has been shown to develop rational solutions to a design problem which enables the design engineer to review the impact of early design constraints. The framework is also flexible enough to enable systems of differing capabilities to be integrated into the framework providing that suitable 'Wrappers' are developed.

1.9. Overview

The topics covered in this chapter, present an introduction to the issues covered in more detail in the following chapters. The next chapter intricately examines the restrictions and issues faced in the design process. Specific examples have been analysed in the domain of the design and selection of pumping systems. Concurrent engineering principles, and there useful application in resolving some of the design problems identified are discussed.

Chapter 2 therefore provides a general justification of and highlights the need for, the development of a computational framework for support engineering design.

Computational mechanisms to support the design process are then outlined in chapter 3. Many different tools and approaches exist for supporting the design process. The methods are explained and their capabilities reviewed. Chapter 4 then develops a theoretical framework to support engineering design. The rationale behind each part of the framework is presented, and what is expected from the framework is discussed. An important part of the framework to support distributed design is the ability to resolve conflict. How conflict is managed and resolved is the subject of the next chapter, chapter 5, and is the main development in this research. Chapter 5 discusses the essence of conflict and how to resolve conflict in both the human and computational forms. The computational forms of conflict resolution developed were based on analogous techniques in the human workgroup.

Chapter 6 then introduces the framework as implemented and describes the rationale behind the implementation approach. A detailed design of the CDEX (Concurrent Design EXpert) framework is provided in Appendix K. The CDEX framework is tested using a real-life engineering design scenario between two engineers from different disciplines. The scenario together with how the different agents were built to represent the two engineers is described in chapter 7. An analysis of the results of applying the framework to the design problem is described in this chapter. Chapter 8 concludes a review of the CDEX mechanism, the results obtained, and discusses future developments and improvements to the mechanism that were deduced from its application to a design problem.

Chapter 2. Industrial Problem

2.1. Overview

The process industry is a highly competitive with many different types of company competing for a share of the market. The different types of player (contractors, client companies, equipment providers) all have different issues and objectives to account for in the design and construction of process plant. It is important to have an understanding of these issues and motivations in order to develop suitable requirements for a framework to support the design process. These players and motivations are discussed in this chapter.

A general overview of the design process of chemical plants is included in this chapter, with more detail provided in Appendix M. In order to identify problems at a detailed level, research was carried out into a selected area of the design process: the selection of pumping equipment. More detailed coverage to this area of the design process is provided in this chapter. This discussion provides significant detail on the cause of design problems, at what stage of the design process they occur, and the reasons why these problems occur. The results of this analysis is used as the basis to emphasise the importance of the concurrent engineering approach towards solving the problems identified. Concurrent engineering is then described with special regard to its applicability to the design of process plant.

2.2. Background

2.2.1 The Players - Client and Contractor

The Client is usually a chemical company (e.g. ICI, BP) that has identified a requirement for a particular product or has been contracted to produce a product for another company. The identification of markets in the process industry is more complex due to the time span involved in getting from a chemical concept to producing a marketable product. A typical plant can take between three and four years to design and construct and market conditions can change considerably during this period. There is therefore a great emphasis placed on reducing the amount of time taken to design and construct a plant. There may be considerable cost benefits to be obtained if a plant is able to produce the product just a few months earlier.

The client is also under pressure to improve the quality of the plant design. Plant failures can be very costly and legislation on emissions to the environment all require improvements throughout the design process.

Traditionally the Client used to be heavily involved in the design and construction of a plant but it is now usual to find a major proportion of the work carried out by a contractor.

The contractor is usually an engineering company that specialises in the design and construction of chemical plants. The contractor itself may employ the services of other contractor companies and individuals for the duration of a project. The contractor is involved throughout the design and construction of a plant and usually has little to do with its operation.

The objectives of the contractor are not far removed from those of the client. The contractor aims to complete the project within time and budget to avoid a loss in personnel profits and to avoid damage to his reputation. A safe plant is also a primary objective for similar reasons as well as problems concerning liability. Secondary to all these objectives are an optimised plant design. Reviewing design for optimisation in energy costs, maintenance costs and construction material costs can be very time consuming and the contractors time to spend on this problem is limited.

2.2.2 Business Organisation

An understanding of the business organisation of the client and contractor is important in understanding the design process. The businesses are organised into functional groups that each perform a task relevant to a particular part of the design.

The Client

The client has a major input into the overall design and operation of the plant but has little to do with the detailed engineering and construction.

The client initially approaches the contractor with a plan of what is required to be designed and constructed. This plan may be as simple as a description of a product that is required or a more detailed process flow diagram. A process flow diagram depicts the major flows and equipment in the process as well as available feedstocks and product.

The contractors initial involvement with the client will usually be with the process engineers who discuss with the client the feasibility of the project and possible problems. The contracting company may own the process and therefore will discuss the requirements and cost of realising the process with the client.

When the contract has been approved, the client will usually appoint a project manager and a small team from his own personnel staff to monitor progress on the project. This team will usually consist of experts from all the different disciplines that may have an input in the project. These experts will monitor the progress and quality of the work done by their appropriate disciplines in the contracting staff.

The client is ultimately responsible for operation of the plant. If the process specification was provided or purchased from the contracting company then usually the contractors will provide some support with operational problems.

The Contractor

The contractor is responsible for designing and building a safe plant to the agreed process specification within an agreed time interval.

The contracting staff consist of all the specialist services that are required throughout the design and construction of a plant. Brown & Root Braun (B&R) setup individual project teams that consist of these various disciplines for each project. There is very little matrix style management where engineers operate across projects.

The detailed design performed by the contractor forms the basis of this research and therefore the tasks performed by the contractor are considered in greater detail here. For the sake of clarity the complete design process from initial chemical path synthesis through to plant construction and commissioning are described. In a real situation the client is likely to have performed some of the earlier tasks.

2.2.3 Stages of Process Plant design

The design process is inherently complex and requires the expertise of many different disciplines. The diagram below depicts the major aspects in the engineering design of process plant:

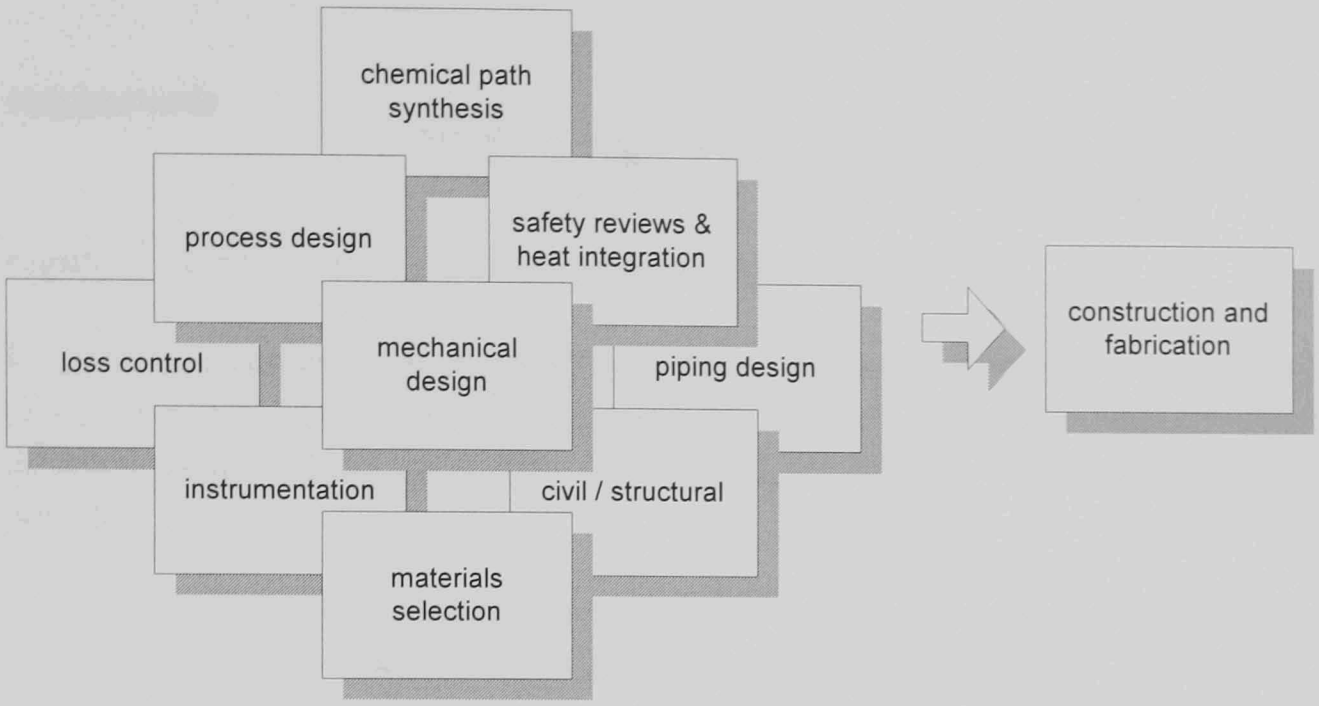


Figure 1 Design tasks in the engineering design of process plant

The design process depicted above is described in Appendix M. Each of the disciplines shown plays an important part of the design of the plant and decisions made by each disciplines inherently effect the decisions made by others. This web of communication channels and dependency makes the management of the design process and design data a highly complex problem. The problems inherent in the traditional approach to engineering design is the subject of the next section.

2.2.4 Problems with the traditional design process

The problems faced in the traditional sequential approach to design and manufacture have been well documented [Tomarchio 91]. Products take too long to develop, cost too much to produce, and often do not perform as promised or expected [Cleetus 92]. The major causes of such problems are the lack of integration between disparate design disciplines, they neither understand nor consider the goals and constraints of other disciplines involved in design. These misunderstandings between disciplines cause significant amounts of design re-work and design compromise.

The engineering design of process plants is highly sequential with design tasks being performed without enough consideration to the final product (Figure 2). Engineers will work towards producing the best design from their own viewpoint before passing their design on to other disciplines for further refinement. This type of engineering generally fails to consider the impact of design decisions on later design refinements and has appropriately been termed 'over the wall' engineering.

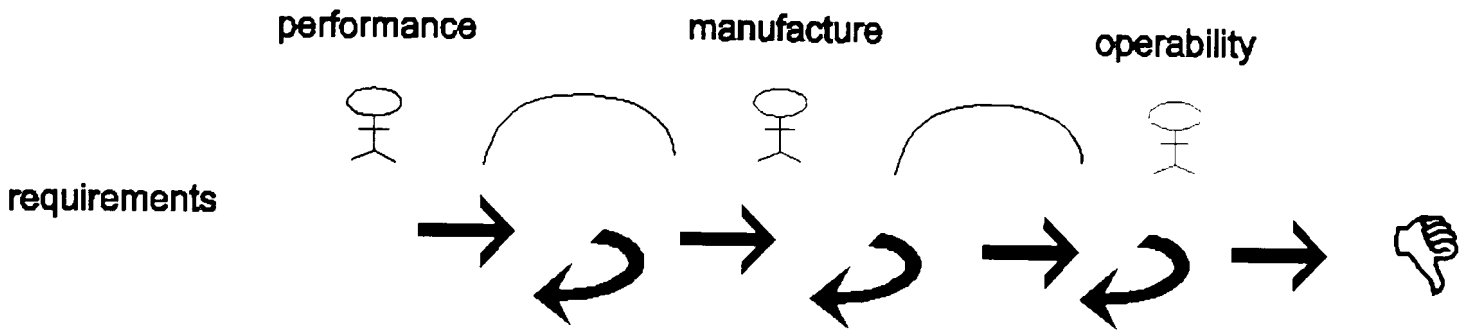


Figure 2. Traditional approach to engineering design

It has been estimated that about 10% of the work carried forward in the design has to be re-iterated [SfK 92d], however, this estimate is likely to deviate between disciplines at different stages of the life cycle. This high level of iteration can be assigned to three reasons;

i. The disciplines involved throughout the design process fail to understand the issues involved in other areas of design.

ii. An engineer does not necesarily have a clear understanding of how their work fits into the complete design cycle and how their task effects the final quality aspects of the plant

iii. An engineer attempts to produce the best possible solution within the confines of their own discipline. This may be in the form of a completely optimised design, a 'clever trick' or may be just a nice design style. These solutions however may be the cause of many problems for other design disciplines.

The traditional design process does not facilitate good communication between the engineering disciplines. This lack of integration and sharing knowledge can lead to repeated problems, as engineers do not receive feedback on their design. An example was identified where a large company had been having a well above average number of fires with its air coolers and the situation was not improving [SfK 92e]. The people who were to specify the equipment were unaware of the operational problems with the equipment and therefore continued specifying faulty compressors. It has been found that some large companies fail to maintain records of equipment failures on its plants so are unable to identify most of the problems. Equipment failures are only likely to come to the engineers attention when there is a serious accident or failure of a plant.

The quality of a design can only really be assessed by reviewing the lifetime history of the plant in operation. The decisions that are made by an engineer are therefore based on 'best practice' - what has been successful in the past and peoples' experiences. In order to improve the design process, strengthening this level of feedback is important.

The more experienced engineers - especially those who have experience in a number of design departments and probably in the field - will produce better designs because they can

bring their broad experience to bear on the design problem. This wide range of experience will enable them to identify and assess the impact of their design decisions on the rest of the design. The intent is to avoid design iteration in the later stages of engineering design when problems can prove costly in terms of project delays. Additionally, the effort involved in design modifications may be prohibitive and therefore design compromises will have to be accepted in order to complete on time. These compromises are likely to impact on the efficiency, safety, and maintainability of the plant.

The way in which the market in the engineering of plant is developing will put a strain on engineering design teams. There is a strong pull to reduce the time taken to design and develop a plant as well as reduce the overall cost. This means utilising as few engineers as possible in the most productive manner. The time constraint will impose more urgency in ensuring that the results from various stages of design are correct first time, while at the same time promoting design compromise in order to deliver on time. In recent years the economic climate has seen a large proportion of experienced, well qualified engineers leave the industry. This experience is not being maintained or captured within the engineering organisation in which they work. Costs have promoted the use of younger, therefore less experienced engineers on projects which is likely to impact on plant quality and safety.

2.3. Pump Selection - a detailed study

2.3.1 Pumping systems design as an example domain

It has long been recognised that the poor design of pumping systems has proved costly for the chemical process industries. As a general statement, pump maintenance can account for up to 20% of the total maintenance cost of a chemical plant, depending on the corrosive nature of the plant and other design factors [Dorma, McKenna 73]. A pump employed in the chemical industry will have a maintenance cost of approximately twice the value of the pump in the first five years of life, but will struggle on for typically 25 years, before being replaced [Barnard 92]. From Barnards [Barnard 92] career in the chemical industry, experience suggests that 90% of the problems associated with pumps (including bearing and seal problems) have their origins in the impeller design and rotor dynamics of the pump. It is suggested that these problems have their roots in industry standards, customers specifications and standards, whims and fancies, for it is the fitting together of components to satisfy a customers specification that can move the operation of any one, or sometimes several of the components away from their true design point.

Research at the BhR[1] group has found that the poor design and selection of pumping systems can predominantly be attributed to a lack of consideration to downstream

---

[1] British Hydromechanics Research Group. This research was carried out on behalf of this company as part of the postgraduate partnership scheme.

activities, as well as poor tradeoffs between the complex goals involved. This lack of consideration leads to non-optimal and compromise design decisions.

2.3.2 Classification of the 'cause' of pumping problems

The following problems with pumping installations have been extracted from research carried out with the British Hydromechanics Research group (BHR) and the relevant literature. The problems have been classified in order to identify the more general problem areas in the design process. These classifications are depicted in Figure 3.

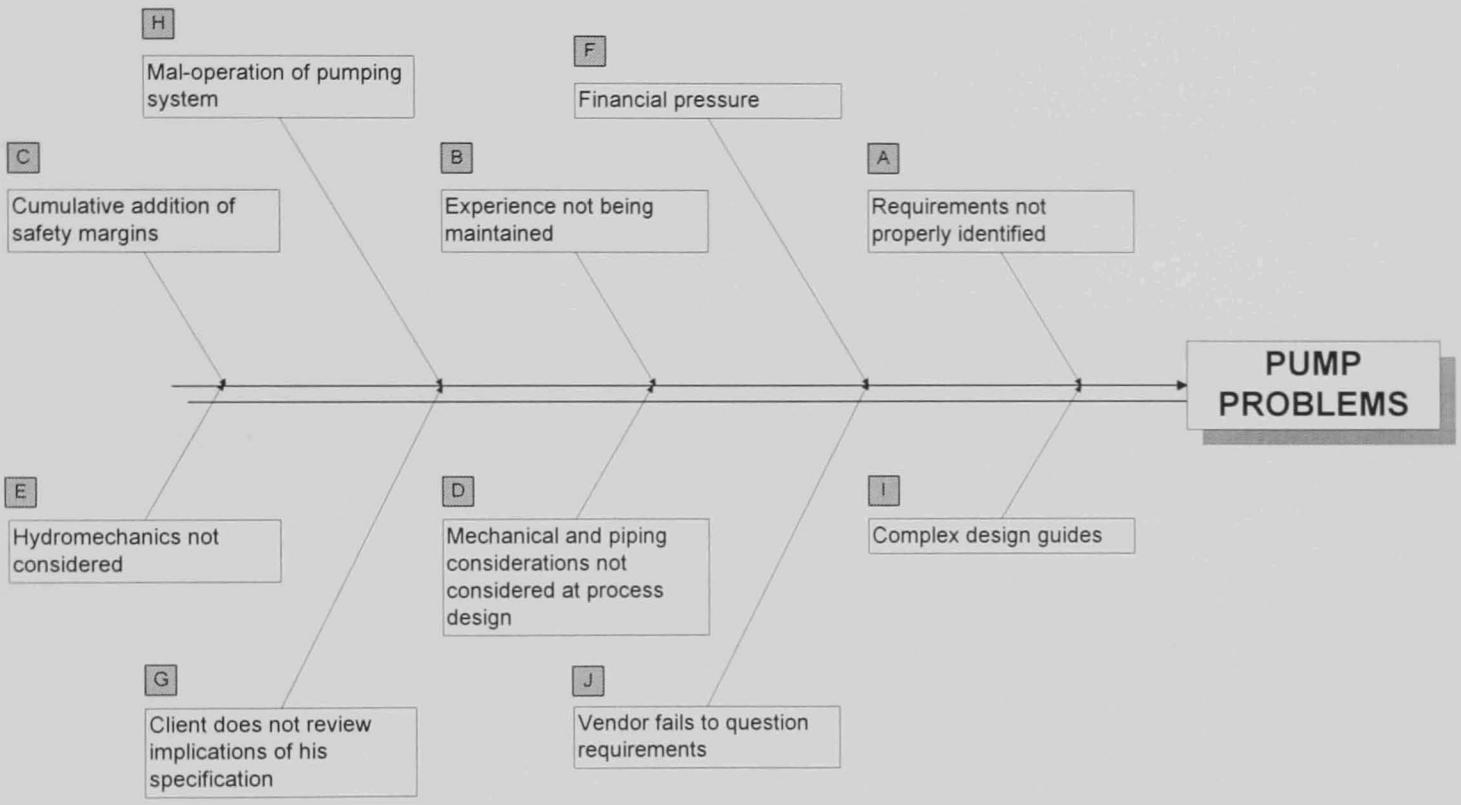


Figure 3 The Generic Causes of Pumping Problems

a. Requirements not properly identified

Many of the problems identified have been related to the improper specification of the requirements. Rather than being due to a generally poor specification (a.8, a.9) they have been attributed to the following factors:

i. The client does not know what he wants (a.1, a.5)

ii. The process engineer does not consider the pump operating parameters (a.3) and range of operations (a.2)

iii. The process engineer does not know what information the vendors require (a.4. a.5. a.7, a.2)

iv. The translation of the process requirement by the mechanical engineer results in the loss of design intent (a.6)

a.1 With operating experience the deficiencies in the original specification are uncovered along with deficiencies in the design of the equipment. Problems arise out of ignorance on behalf of the customer as to what he really wants, and what information is important to the vendor for his design and vise versa. A vendor may often supply a piece of equipment for what he thinks is necessary to do the job. often with ignorance of the application, the service conditions, and often failing to ask for further details of the application. [Barnard 92]

a.2 It was thought that the best way of tackling the problems with pumping installations would be to ensure that the process engineer produced good process data sheets and general arrangement drawings [SfK 92b].

To properly specify the process requirements for a pump, the design engineer must explore the full range of operations the pump will be expected to perform. Considering only the normal operating case could lead to disappointing performance at either higher or lower-than-normal flowrates [Fischer 87].

a.3 Many problems are derived from the fact that the process engineer does not always take pump operating parameters into account when he's designing the process. By the time the mechanical engineer or vendor spots the potential problems the design is already cast in stone. [SfK 92c]

a.4 An SfK study [SfK 94] has classified the pump problems on site into poor manufacture, poor operation, and there are times when the pumps are specified incorrectly as the process engineers do not know the requirements of the pump vendor [SfK 94].

a.5 Problems arise out of ignorance on behalf of the customer as to what he really wants, and what information is important to the vendor for his design [Barnard 92].

a.6 The mechanical engineer uses a precis of the process engineers' work to formulate a specification sheet. Much of the process design intent is lost in the translation. [SfK 92c]. In some cases the process engineer would be quite happy to see the mechanical engineers bypassed completely.

a.7 From the vendors point of view many of the problems occur because they do not have access to the right level of detail [SfK 92c].

a.8 The system curve is not shown on the pump data sheet API610. This is a major restriction and leads the vendors to over-specify in order to allow for contingencies [SfK 92c].

a.9 Ingersoll Rand has identified that 80% of pumps on a refinery will have a power requirement of less than 10 KW. It is these pumps, which cause the vast majority of reliability problems, since they do not receive much attention when they are specified. [SfK 92c].

b. Experience not being maintained

A number of problems have been identified due to a lack of experience. These are mainly due to the industry losing talented personnel (b.1, b.2) and a lack of feedback from operations to the design team (b.3, b.4, b.5).

b.1 Many problems exhibited in the past are reappearing as designers try to cut down on margins and as expertise is lost from the industry [SfK 92c].

b.2 Average age of design engineers is reducing and ICI are losing their most experienced personnel. Also contractors do not have experience of running the plants [SfK 92b].

b.3 Many repeat problems by machines section (e.g. wrong lubrication arrangements on paired bearings [SfK 92b])

b.4 Most effort goes on correcting problems, rather than ensuring that they do not happen again. Also the problems with pump operation are not fed back to the design team. Major problems are, but minor problems (bearing, seals etc) are handled on the plant [SfK 92b].

b.5 Plant maintenance being carried out by workshops rather than plant maintenance teams. Important operational expertise is being lost [SfK 92b]. This will get worse as workshops are contracted out.

c. Cumulative addition of safety margins

c.1 Pumps do not operate at their duty point when new. Process engineers add safety margins into their calculations, as do the machines engineers. The cumulative effect being that the pump does not run at BEP [SfK 92b].

c.2 ICI have identified problems with pumps, mainly with the seals and bearings failing. ICI were not surprised as pumps are never run at their design point anyway. [SfK 92b]

c.3 Pump reliability problems in the vast majority of cases occur due to two main factors [SfK 92c]:

It was estimated that around 90% of pumps used on peripheral services are over specified in terms of head.

Many pumps are operated at duties below their recommended minimum operating range.

d. Mechanical & Piping considerations not considered at process design phase

d.1 Information passed between the process, mechanical and piping is always the ideal solution that often cannot be met exactly and therefore compromise solutions have to be made [SfK 92b].

e. Hydromechanics not considered

e.1 Piping engineers produce isometrics (in league with civil engineers) and layouts. These are considered in a project team although it is not normal to consider the hydromechanics [SfK 92b].

f. Financial pressure

f.1 Many problems exhibited in the past are reappearing as designers try to cut down on margins and as expertise is lost from the industry [SfK 92c]. Because of this, pump reliability is declining rather than improving, and many of the mistakes are elementary text book errors. Some new problems are occurring though, such as acoustic related problems due to lighter pipes and faster pump combinations.

f.2 ICI commented that it is quite common to freeze designs even though they are not ideal when the cost of re-working the design would be prohibitive [SfK 92b].

g. Client does not review implication of his specification

g.1 From the authors' career in the chemical industry (Paul Barnard -Exxon), experience suggests that 90% of the problems associated with pumps, and that includes bearing and seal problems, have their origins in the impeller design and rotor dynamics of the pump. It is suggested that these problems have their roots in industry standards, customers' specifications and standards, whims and fancies, for it is the fitting together of components to satisfy a customers specification that can move the operation of any one, or sometimes several of the components away from their true design point [Barnard 92].

h. Mal-operation of pumping system

h.1 It was unclear as to how many pump problems stemmed from bad design. in many cases bad operational practices were known to cause the problems [SfK 92b].

I. Complex design guides

I.1 Contractors do not follow the design guides [SfK 92b]. The design guides are complicated and require that you know quite a lot about pumps to make best use of them. There too much for most people to use, people either do not read them. or don't understand the implications of what they are reading.

j. Vendor fails to question requirements

j.1 A vendor will supply a piece of equipment for what he thinks is necessary to do the job, often with ignorance of the application and service conditions. and often failing to ask for further details of the application [Barnard 92].

2.3.3 Why do we have pumping problems?

It can be seen from the above analysis that many of the above problems can be tackled with more discussion between the different design groups about what is required, what the safety margins are, and bringing back operational experience into the early design stages. The following figure depicts the tasks involved in pump system design throughout the complete design life cycle. The problems characterised above have been highlighted in figure 4. It shows that many problems occur in the later design phases where of course designs are subject to compromise as to go back and repeat an early design task is expensive in time and effort. The characterised problems shown in the early design phase indicates a need for the process engineer to review the design from the life-cycle perspective.

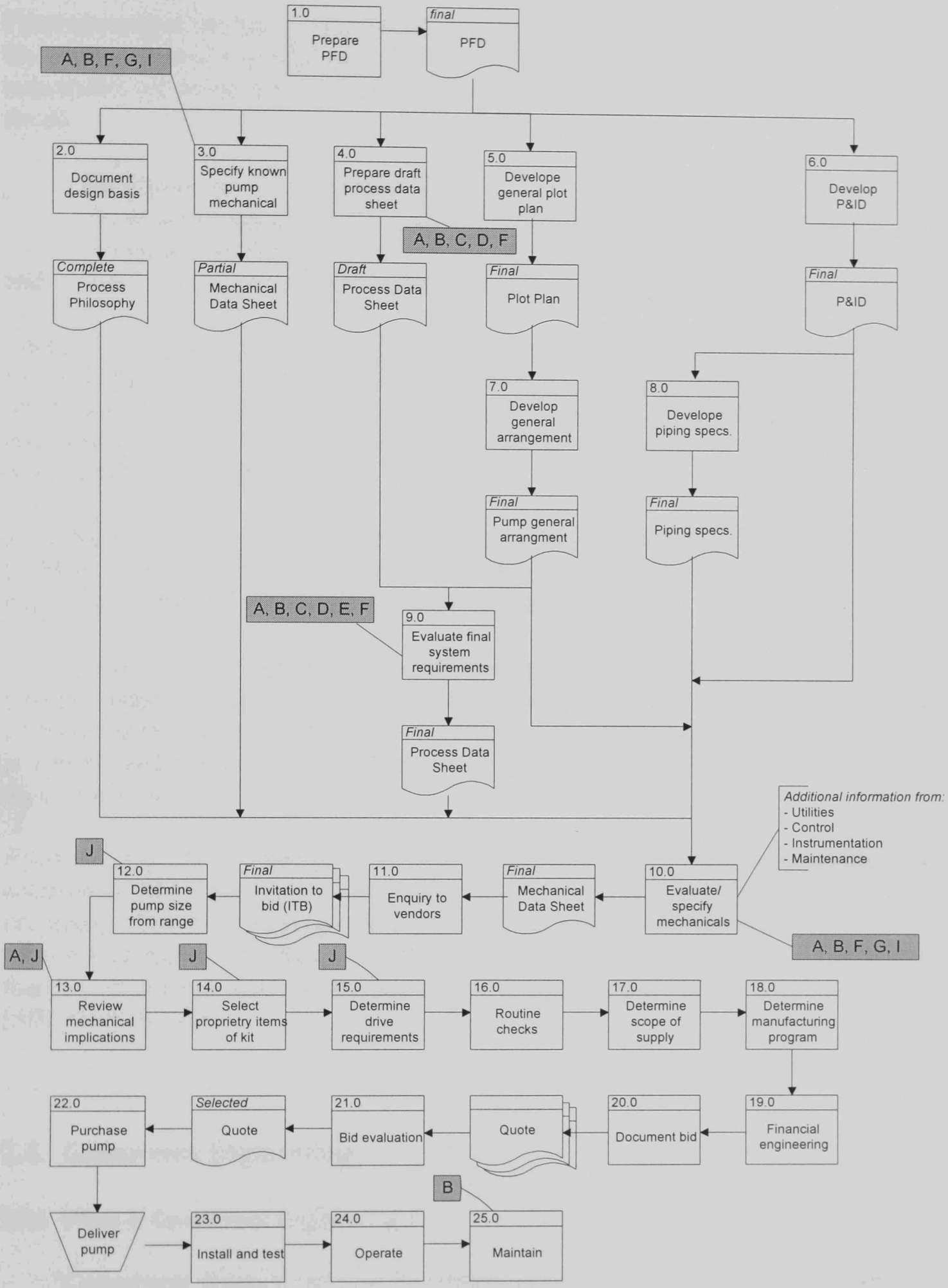


Figure 4 Tasks Involved in Pump Selection and the generic cause of Pump Problems

2.3.4 Review

From a review of the design tasks and associated problem classifications, a tool to support the process engineer would seem of most benefit. The process engineer is primarily responsible for defining the *requirements* where the following problems were generally found:

the client does not know what he wants
the process engineer does not properly consider the range of operations
the process engineer does not know what information the vendors require
and there is a loss of design intent between the process and mechanical engineers

Finding problems at the requirements definition stage is not surprising given that the decisions made early in the design life cycle are usually 'larger grained' - i.e. have a greater impact on the overall result. The early decisions set the lower bounds on cost, time complexity and risk, as well as establishing the upper bounds on achievable product reliability [Cleetus 92].

A problem that the industry is facing is how to improve the situation. Currently the problem appears to be getting worse rather than improving. This is due to a lack of experience being maintained, and financial pressures.

The industry is losing key people, and the average age of the design engineer is getting younger. This is leading to an overall loss of corporate *experience*. There are many repeat problems as mechanical engineers are not made aware of the operating problems. There is also no feedback to the design teams -especially now that contractors are performing most of the work [SfK 92b][SfK 92c].

Financial pressures are leading to the use of younger and therefore cheaper engineers by contractors. Many problems are elementary text book errors, although new problems are occurring. The use of cheaper, thinner pipes has lead to new acoustic problems and vibration problems. It was also noted that it is quite common to freeze a design even though they are not ideal because the cost of re-working a design would be prohibitive [SfK 92b][SfK 92c].

2.4. Concurrent Engineering

2.4.1 What is Concurrent Engineering ?

> *"Concurrent Engineering can be termed as the systematic approach to the concurrent design of products and their related processes, so that considerable reductions in time and cost can be achieved"* [Gopalakrishnan, Pandiarajan 90]

> *"**Concurrent Engineering** is a **systematic approach** to integrated product development that emphasises **response to customer expectations** and embodies **team values** of cooperation, trust and sharing in such a manner that **decision making** proceeds with large intervals of **parallel working** by all life-cycle perspectives early in the process, **synchronised** by comparatively brief exchanges to produce **consensus"*** [Cleetus 92]
>
> *"Integrated engineering addresses the entire way we design and develop products. It is a simple theory that aims to achieve the optimum design by ensuring that all the engineers understand the limitations involved in the design and manufacturing process and account for these limitations in their work"* [Reklaitis, Preston 89]

Integrated Engineering is referred to above as a preference to Concurrent Engineering. Integrated Engineering, Concurrent Engineering, and Design for Manufacture are all based essentially on the same principles and the terms are often used inter-changeably.

Concurrent engineering promotes a teamwork approach to design where the team is responsible for ensuring the decisions made early in the design do not have a detrimental impact on later design stages (Figure 5). The team is made up of engineers involved in each stage of design (a multi-disciplined team) so they bring their knowledge and design viewpoint into consideration. This approach reduces the problems usually found in the later design stages therefore reducing iteration and design compromise. The approach has proven successes [Knodle 91][Dutcher 91][Tomarchio 91].

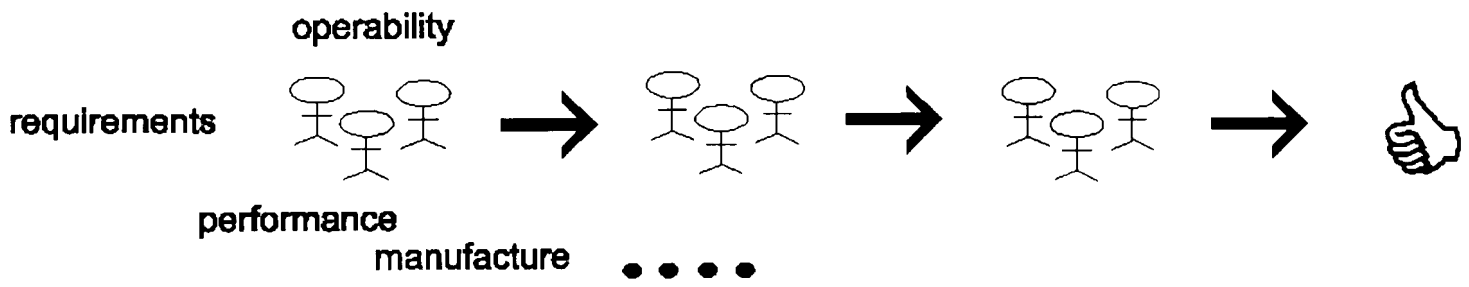


Figure 5 Concurrent engineering approach to design

The following is a list of concurrent engineering practices: [Knodle 91].

i. Proactive, end item responsible, product development teams
ii. Parallel development processes
iii. Expanded product requirements, documents to focus on life-cycle, requirements and cost
iv. Integrated activity schedule
v. Life cycle flow chart depicting entire hardware development path
vi. Performance metrics for the team, processes and project

vii. Tailoring of all functional outputs to meet "internal customer" needs
viii. Proactive usage of design for manufacturability and assembly methods
ix. Proofing of all processes prior to production implementation
x. Utilisation of concurrent engineering tool where applicable:

Computer Aided drafting and engineering (CAD/CAE)
Quality function deployment
Taguchi Methods
Statistical process control methods
Simulation tools for mechanical and electrical systems

The need for this parallel working early in the project is emphasised through an important observation: fuller consideration has to be given to the larger grained decisions made initially because they set lower bounds on cost, time, complexity and risk, as well as establishing upper bounds on achievable product reliability and customer satisfaction [Cleetus 92].

There are many ways to describe the aims of concurrent engineering. The goal in the process industry is to build plants with optimised capital cost, efficient operation and built in shorter time frames. This requires not only that the tasks of the product realisation processes be highly and effectively integrated, but that the tasks must proceed in parallel to as high a degree as possible [Reklaitis, Preston 89].

A simple guiding principle is to strive to get a job done **'right first time'**. Substantial savings in resources can be achieved if several operations, including preliminary design, can proceed in parallel. This implies that some of the operations occur concurrently, while some other function in an advisory mode [Gopalakrishnan, Pandiarajan 90].

2.4.2 Concurrent Engineering in practice

The main emphasis of integrated engineering is to build multi-disciplined teams that are responsible for designing and developing a product. This team can critique a design from multiple task perspectives.

This team will reduce the amount of iteration between the various design stages as problems will be identified earlier and the appropriate corrections made. The design should progress through all the design stages without any major complications. Studies have shown that correcting a defect found late in the product development cycle can cost ten times more than identifying the problem at an earlier stage [Knutton 92][Tomarchio 91]. British aerospace studies have also shown that by the time 5% of the product development time and cost had been expended, 85% of the product life cycle cost had been committed.

MacDonell Douglas have had extensive experience with multi-disciplined product teams [Dutcher 91]. They have set up core product teams that are collectively responsible for the design. The team is not a static entity, people leave and join the team, including the team leader, depending on what stage the product has reached. They found it best that the team leader is educated in a variety of disciplines but is a specialist at the task in hand. The team leader has to act as the arbitration mechanism if conflicts arise [Tomarchio 91].

Special care is required in the setup of core teams. Due to a large number of disciplines needed to support a project, special attention must be given to the coordination of information and the size of the team.

2.4.3 Difficulties in concurrent engineering

Knowledge at varying levels of detail

Difficulties in concurrent engineering are encountered when large multi-disciplinary projects are involved where the team members work at different levels of detail, using different models and different knowledge. Computational support for concurrent engineering aims to develop tools to help team members in sharing knowledge and keep track of the others' needs, constraints, decisions and assumptions [Cutkosky, et.al. 93]. Such systems would enable engineering disciplines from each of the design life-cycle stages to communicate and review design strategy. As a group they would be able to explore design alternatives in search of a good solution [Talukdar, Fenves 89].

Complex Interdependencies and Parallel Working

In complex design problems like the engineering of plant, there is a host of complex inter-dependencies between the parallel design tasks. The human ability to adequately comprehend and evaluate design alternatives diminishes with the increasing complexity of these inter-dependencies [Cutkosky, Conru, Lee 94][Anson, Jelassi 89]. Simmons [Simmons 93] identified that a sub-conscious assessment of relevant design information is unacceptable given the increasingly complex technology and the increasing consequences of wrong decisions. It is clear that there is scope for improving design decisions by providing the decision maker with advice on the probable consequences of his decisions [Simmons 93].

Parallel working has the inevitable penalty of inconsistency after some time, because the partitioning of tasks along roles is approximate in the best of situations; an overlap often remains. This leads to conflict among the individual decisions as the parallel tasks progress in time. The resolution of these conflicts has to be planned, and this is done by announcing a point of synchronisation every so often so that the emerging alternatives and details of the decisions may surface to the entire groups view. The team decides how to trade off among alternatives and render decisions consistent again before the next regime

of parallel tasks is set in motion. The thesis is that these cycles of convergence may be more numerous in the life of a project than the find and fix cycles of the past, but the time required to achieve consensus is less since the minds meet more often over smaller divergences of viewpoint. Coordinated working among team members demands frequent exchange of information so that the impact of decisions made at one stage is not allowed to lie unexamined till late in the project by those whom it might effect [Cleetus 92].

Organisational change

The main problem found in industry is breaking down the procedural and physical brick walls that exist between the various disciplines. In order to solve this problem the commitment of the management and people is imperative to the success of a project.

There are two types of change in an organisation that can be classified into [Tranfield, Smith 90];

i. Morphostatis

   This method of change is incremental. Change usually occurs in small steps and is usually due to the effect of disturbances.

ii. Morphogenesis

   This is transformational change. This change usually occurs after a review of where the company is going and the setting of objectives.

There are problems with both types of change. Morphostatis change can lead to the building of small empires of specialist teams. These teams set their own standards and may not be working to the same goal as the company. Morphogenic change is more radical. Morphogenic change leads to a feeling of insecurity and therefore workers tend to be resistant to change. Managers tend to become embedded in the system and therefore are less likely to implement morphogenic change.

Morphogenic change is necessary to successfully implement integrated engineering and therefore total commitment of the management and the workforce is necessary.

Although problems have been identified with integrated engineering, it appears the right way forward. Any company attempting to implement integrated engineering practices is advised to start with a pilot study [Costanzo 92] and give very careful consideration to the development of a team. Initially Macdonnel Douglas had the problem of breaking down the procedural and physical brick walls that existed between the various disciplines. These problems are similar in most industries and therefore the commitment of the management and people is imperative for the success of a project.

Generalisation of skills

From a process industry perspective, ICI have also had problems to overcome when adopting a teamwork approach to design. A project was managed by a committee that was made up of experts from the various disciplines. There were also a variety of sub-groups that were responsible for carrying out the objectives defined by the committee. A number of problems were identified with the approach. When problems are encountered by the group they were reluctant to pass the problem over to the committee because they thought they could deal with it. Also the sub-groups did not always realise that a problem existed and therefore the committee was not aware of it. Another problem that was considered more serious was the de-skilling of the workforce. If a group was set up from a variety of disciplines, their skills subsequently became more generalised. The conventional design methods of grouping experts within a particular specialist department lead to engineers concentrating on a single subject and learning from their partners. With the introduction of teams, this was no longer the case.

ICI have already noticed a decrease in the number of experts within the company and they consider that by employing teamwork methods they may be exacerbating the problem. ICI have posed the big question, "How do they grow their new expert ?" [SfK 92a].

Any strategy developed to implement integrated engineering practices must embrace all the product development disciplines to be effective. The strategy must also be adaptive and ever changing to suit the needs of the environment.

2.4.4 Potential of concurrent engineering in resolving design problems

The trend in the engineering design sector is to improve the safety of process plant, and to design, build and operate them in the most cost effective manner. This requires not only that the tasks of the product realisation processes be highly and effectively integrated, but the tasks must run concurrently to as high a degree as possible [Reklaitis, Preston 89]. These time constraints impose more urgency in ensuring the design is **right first time** and that there is less design compromise. This change is inevitable, the process industry faces changing global competition and legislation which requires companies to be ever more efficient and cost effective to survive.

In order to achieve these goals, the traditional sequential approach to design is not adequate. Engineering disciplines need to communicate and resolve design issues early in the design. Concurrent engineering is a method in which an engineering design team comes together in order to communicate their ideas and resolve early design issues. Computational support is required however where there is a large number of complex and interacting design issues. This support will inherently be knowledge based due to the considerable knowledge required in supporting engineering design decisions. At the same time the aim is to utilise current design technologies (e.g. CAD, process flowsheeting packages, sizing programs).

2.5. Summary

In order to reduce product lead times, improve quality and ensure proper use of computing technologies there must be change in the business organisation. Changing the structure and culture of organisations is a very difficult task and cannot be performed without full commitment from the management and the workforce. This chapter has described the latest methods in industry, known as integrated or concurrent engineering, and why they should be adopted by the process industries. Change throughout the process industry is inevitable. The changing face of global competition requires companies to be ever more efficient, safe and cost effective to survive. The chapter discusses the rationale behind bringing computational support to the design of process plant. The engineering design process of chemical plant together with a review of the motivations involved was described. Problems in the pumping domain were targeted for specific analysis in order to identify the root causes for problems in sufficient detail. A lack of communication between the design disciplines was found to be a problem. Research at the BhR group has found that the poor design and selection of pumping systems can predominantly be attributed to a lack of consideration to downstream activities, as well as poor tradeoffs between the complex goals involved. This lack of consideration leads to non-optimal and compromise design decisions.

These problems can be ironed out through improving communications between the different design groups and working together. Concurrent engineering is described which is an approach to enable effective team working and resolves some of the problems identified. It is clear from this study that the traditional sequential approach to engineering design is not adequate and a more advanced approach that includes an element of team working is required. From a review of the associated problems and associated stages of the design where these problems occur, it can be seen that a tool to support the process engineer in assessing design alternatives would seem of most benefit. The decisions made by the process engineer are also 'larger grained' - i.e. they have a greater effect on how the design will progress as compared to those decisions made late in the design phase.

The next chapter provides a review of computational techniques to support the design process to alleviate some of the problems identified. From the understanding of the design process covered in this chapter, the principles for computational support mechanisms to aid the design process are described.

Chapter 3. Computational Support in Engineering

3.1. Overview

In this chapter the rationale behind the engineering design procress is described, together with the requirements for computational support in the design task. A review of the benefits of computational support is provided. As the technologies are becoming more complex, and engineering standards more voluminous, one finds a greater need for knowledge based computational support in the design task. The features required of a tool to support CE are described - from both a design requirement and computational viewpoint. The tools and technologies currently available to support CE are also outlined, together with the issues in the development of a framework to support the CE design process (e.g. data handling and representation issues).

3.2. Rationale for computational support

3.2.1 Why do we need computational support ?

The reasons for the development of intelligent systems can be summarised as follows:

i. Information flows consume the time of the personnel involved [Rychener 88]

ii. Engineers spend more than half their time on managerial rather than technical issues [Rychener 88]

iii. Provision of technical advice [SfK 92c]

> *"In many branches of engineering the technology is becoming increasingly complex and, at the same time, the consequences of making a wrong decision are becoming greater. In these situations to make decisions solely on the basis of experienced judgement is unacceptable. To make a decision on such a basis is to make a subconscious assessment of the relevant information. Surely it is better to make a conscious assessment, since this will necessitate exposing the available information and the logic behind the decision to possible criticism, which can only be to the good...There is little doubt that there is considerable scope to improve the quality of design decisions by providing the decision maker with better information on the probable consequences of decisions."* [Simmons 93]

Many problems exhibited in the past are reappearing as designers try to cut down on margins and as expertise is lost from the industry [SfK 92c]. Because of this, pump reliability is declining rather than improving, and many of the mistakes are elementary text book errors. Some new problems are occurring though, such as

acoustic related problems due to lighter pipes and faster pump combinations.

Average age of design engineers is reducing and ICI are losing there most experienced personnel. Also contractors do not have experience of running the plants [SfK 92b].

Improving the reliability of pumping installations could best be tackled by providing tools to the process engineer [SfK 92b].

iv. The problems posed by constraints in the design phase are difficult to consider in later design stages [Rychener 88].

Ingersoll Rand has identified that 80% of pumps on a refinery will have a power requirement of less than 10KW. It is these pumps which cause the vast majority of reliability problems. This is due to their considered low importance in comparison with the large and more expensive pumps, and therefore they do not receive much attention when they are specified [SfK 92b].

v. Design complexity and enforcement of complex standards

Contractors do not often follow the design guides [SfK 92b]. The design guides are complicated and require that you know quite a lot about pumps to make best use of them. There is too much for most people to use, people either do not read them, or do not understand the implications of what they are reading.

Design guides attempt to be non-ambiguous, and therefore they are generally more complex [SfK 92b]. A company is likely to be liable if they are misinterpreted and something goes wrong.

Cutkosky identified that the ability to adequately comprehend and evaluate the design alternatives diminishes with the increasing complexity of the inter-dependencies between the parallel design tasks. This is a significant problem given the additional requirements of concurrent engineering to correlate and resolve life-cycle issues.

vi. The typical design cycle from product conception through to production is time consuming [Rychener 88]

3.2.2 How can computational support benefit ?

Much research has gone into the development of IKBS algorithms to aid in the design process [Oliff 88, Yoshikawa, Gossard 89]. Process design can benefit from these developments in a number of ways:

i. Rapid checking of preliminary design concepts and avoiding design errors, iteration and compromise - *"The Concurrent Engineering Benefit"*

ii. Strategy for iteration over the design process to improve on previous attempts

iii. Assistance/Automation of complex sub-steps of the design process - *"Engineering is about making decisions - not calculations"*

iv. Strategy for searching in the space of alternative designs

v. Ability to integrate design tools therefore reducing transcription error, communication, and engineering effort.

vi. Maintenance of expertise and standard design guides thereby improving quality

3.3. Features of a Computational Support tool for CE

There is a wide variety of technologies that can support CE and to varying degrees. These systems may be from anything as simple as a mailing system to ensure everyone is kept up to date with design progress, to more elaborate and complex knowledge based and design simulation systems. Outlined below are some of the general and more specific features of software that can be said to support a concurrent engineering philosophy. The problems and issues surrounding these features are outlined later in the chapter.

3.3.1 Generic Features

A specification of a system to support engineering design must conform to a set of requirements. These requirements have been identified from a review of traditional engineering design practice and from the concurrent engineering practices which have been identified as beneficial in the design of process plant.

The five generic principles are shown in the following figure, and the more specific features are then discussed.

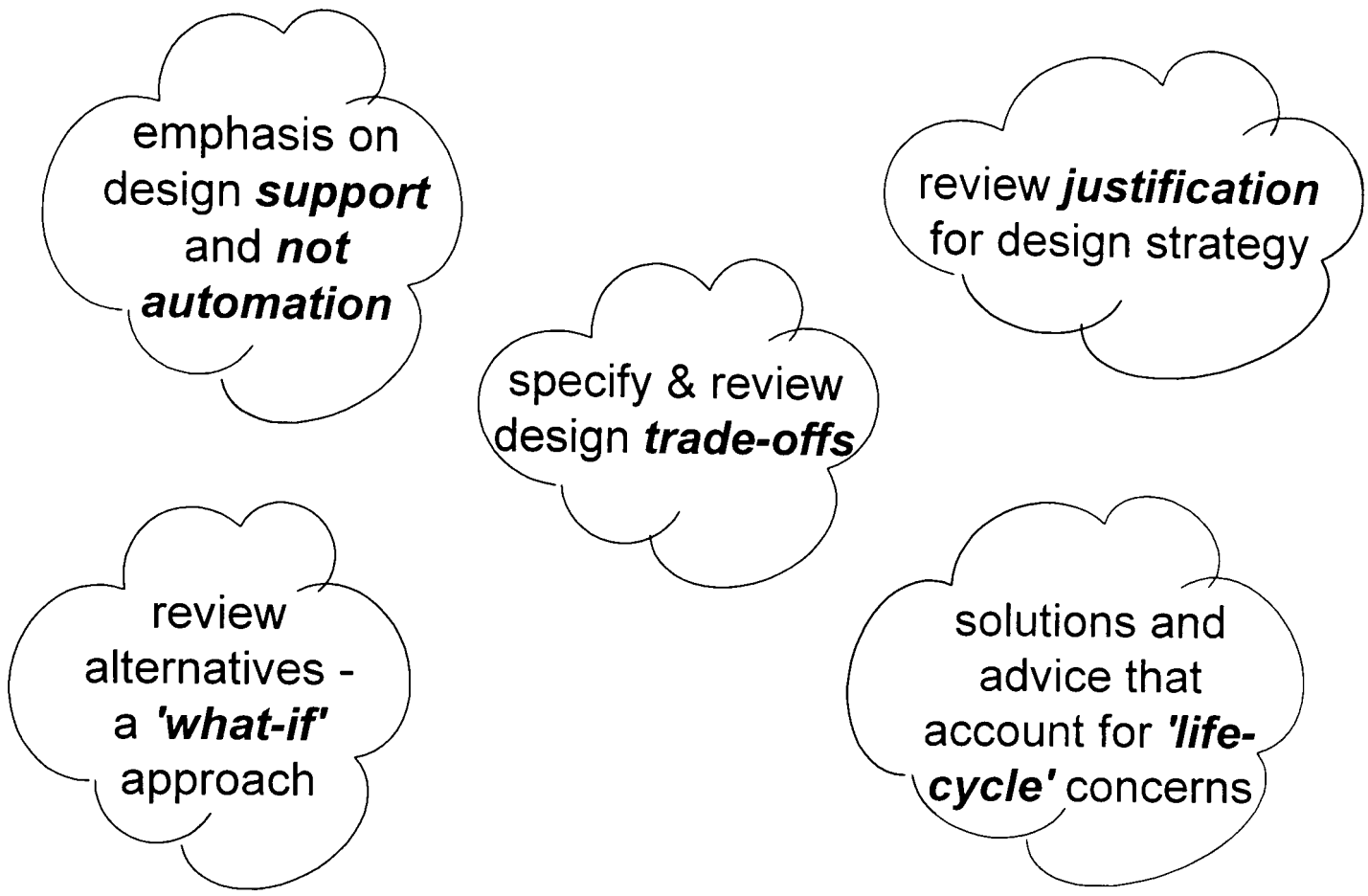


Figure 6 Five of the guiding principles for knowledge based life cycle support for CE

Impact Assessment

Engineers need to access the impact, and identify the constraints that their design has on other design disciplines.

The ability to propagate changes for a continually evolving design is the most important attribute of a system for concurrent design [Brown, Cutkosky, Tenenbaum 89]. Through propagating these changes, the early design effects and constraints on future design decisions can be reviewed.

What-If Analysis

A 'what-if' analysis provides the ability to review other options and determine why those solutions are less appropriate than the one identified. It is important to be able to review the justification for a design strategy. If a system proposes certain solutions to the engineer, then it is important that the engineer can review how the proposal was arrived at. It is wise to present results in a logical fashion so that the engineer can comprehend the transition from initial requirements through to solution. A danger is that the engineer may not understand how the system has come up with its result and what factors were accounted for, therefore having little confidence in any proposals made.

Design Support - not automation

Design support is provided by enabling the engineer to specify hard (in-flexible) and soft (flexible) constraints, and attributes that are fixed and cannot be modified by other engineers in the design process.

It is important to maintain a principle of design support and not automation. The design process cannot be completely automated and nor should one attempt to achieve this goal [MacCallum 90]. There are many variables that effect a design and many views from customers, legal people, varying engineering practices and so forth. There are many factors crucial to the design process that a computer cannot be expected to consider. An engineer should be able to utilise knowledge in the design process that may not be available in the framework (outside the bounds of the system). It is likely that at various stages of the development process of a concurrent engineering tool there will be a lack of computational design knowledge given the dimensions of the design task.

Life cycle review

The engineer should be treated as a highly time constrained resource. Conflicts identified between the different goals and design criteria should be clearly reviewed by the framework before presenting them to the engineer for consideration.

In the life cycle design process, different disciplines have different priorities and goals which they trade off against each other in formulating their part of the design. In a distributed knowledge based system this different criteria should be accounted for in evaluating design solutions. The engineer should have the ability to specify the different trade-offs required by the system that matches his requirements. If problems arise that cannot be resolved due to these tradeoffs then he may consult with other engineers or modify his own value structures

Participation from other engineers

An engineer must have the ability to request information/advice from other engineers.

Engineers involved in the later stages of design can review the trade-offs and compromises made with respect to their concerns in the earlier design phases. It should be possible for engineers to document their reasons (rationale) as to why a particular design route was taken.

3.3.2 Specific technological requirements

Additional to the requirements identified above, there are a number of considerations to account for in computational, development, and useability terms.

Distributed architecture

Software implemented in such a framework should have a high degree of autonomy to enable the application of different reasoning processes and design algorithms. There is no single algorithm appropriate for all design problems and therefore flexibility in developing software with different capabilities is an essential facet of the environment. There is also the additional computational burden of executing this encompassing design tool on a single processor machine.

The traditional knowledge based approach of developing a single, consistent knowledge base to provide the life cycle perspective poses significant problems. Although designing a knowledge based system with a single knowledge base is certainly easier than integrating multiple knowledge bases [Steer, et.al. 93], the problems posed for knowledge elicitation, maintenance, and the addition of new knowledge are formidable. It has been stated that "it may be much easier to coordinate the activities of 20 medium sized machines than to build a single machine that is 20 times as large" [Uma, et.al. 93].

Reasonable 'on-line' response

The software on-line to the user should be able to respond in an adequate time interval. This requirement will ensure that the computer does not spend an infinite amount of time searching for design solutions where a solution may not exist. The computational power from a single processor system is in question, even though in recent years dramatic improvements have been made in computational speed. Hillis puts forward a strong argument on the problem of efficiency from a single processor:

> *"One reason why computers are slow is that their hardware is used extremely inefficiently. The von Neumann architecture has memory on the one side and processing on the other. In a large von Neumann architecture almost none of its billion or so transistors do any useful processing at any one instant. Almost all the transistors are in the memory section of the machine, and only a few of those memory locations are accessed at any one time. The two part architecture keeps the silicon devoted to processing wonderfully busy, but this is only two or three percent of the silicon area. The other 97% sits idle."* [Hillis 85]

Incremental development

The development and maintenance of the framework should be incremental given the possible size of a framework that encompasses the complete life-cycle perspective. Additionally, the systems operating within the framework should be easily maintainable by a number of engineering experts. A decoupling of the modules (i.e. distributed) is most appropriate as opposed to a single large system.

Knowledge Based

Cutkosky [Cutkosky, Conru, Lee 94] identified that the ability to adequately comprehend and evaluate the design alternatives diminishes with the increasing complexity of the inter-dependencies between the parallel design tasks. This is a significant problem given the additional requirements of concurrent engineering to correlate and resolve life-cycle issues. A system that can help us to make decisions, and trade-off the various design parameters is inherently knowledge based.

Integrate with other systems

To aid the engineering designers there has been the development of a formidable array of engineering design tools such as CAD, Finite Element Analysis Tools, Simulators, and knowledge based design tools. These tools have been developed in the confines of individual disciplines and therefore give no consideration to other aspects of design. Over recent years however considerable effort has been spent on the integration of design tools in organisations through the development of common interfaces and data exchange standards.

An argument to support integration put forward by Mandiau and Millot is that *"methods used by an intelligent system to reason about the actions of other systems can also be used to reason with other environmentally non-intelligent dynamic processes"* [Mandiau, Millot 92]. In a system to support engineering design, there is a requirement to integrate external sources which are not intelligent - such as a examples mentioned above, but one also has the requirement to integrate with a more intelligent external source, the engineers themselves. Arguments and results presented would be more acceptable if a particular line of reasoning appropriate to this external source could be found.

Cooperation with other systems - resolving 'conflict'

Concurrent engineering imposes the requirement that all appropriate viewpoints have been considered. This requires a frequent exchange of information so that the impact of decisions made at one stage of the design is not allowed to lie un-examined till late in the project by those whom it might effect [Cleetus 92].

Integrating knowledge based systems poses more problems than traditional systems. Traditional systems require a sharing of protocols and a common data representation. Further research in this area is being tackled under the STEP umbrella which enables different software programs to exchange information. This enables one system, to "understand" the output from another. There is however an additional requirement for an intelligent system to support engineering, as opposed to being able to understand the output from another KBS. This is the ability to form a different opinion, and disagree with a strategy or proposal made. This is where conflict occurs. If one is to develop a system to provide some concurrent engineering design support, an approach is needed to articulate

a "shared" response to a particular problem that accounts for the distributed knowledge available. This would behave in a similar fashion to the human workgroup.

In order to achieve a cohesive shared design an intelligent system may want to critique a design proposed by another system, or receive and evaluate feedback from other agents [Sycara 89].

It is expected that in an individual closed world (single software system), the values for any particular design attribute should not be in conflict. For example, a pump casing material attribute would not be both mild steel or stainless steel at the same time.

When one looks at a distributed domain however, it is possible for an attribute to have two values and both be valid. This is possible where any single software system finds itself in a different *context* from another which affects the value of an attribute. For example:

> *A person from a star would consider that a person on earth is looking into the future because the person on earth can see events before any person on a star. The person on earth however would consider that he is seeing history, because he has seen it after it has happened (albeit only a small delay). Both views are consistent from the context in which the different people find themselves (a separation based on time). When taken from a global combined world viewpoint we can accept inconsistencies as the results from the different people (or agents) are both acceptable.*

From reviewing this case, inconsistencies cannot be accepted within a single agent, but from a global perspective, inconsistencies are valid (in fact you would not call them inconsistencies). This is primarily because the agents are operating in a single context, not a world context -therefore world contexts have to include the contexts of different agents.

Manage and maintain design consistency

Distributed environments present particular problems concerning maintenance of design consistency. The distributed software systems may make different assumptions, may not present the relevant information and assumptions to other systems in the design process, record and work with out of data information and a host of other problems. Throughout the design process much data is generated, and this data is often sequentially dependent on other design information [Motard 89]. The status (level of confirmation) of a data item and its quality (confidence) needs to be recorded.

Handle uncertain information

In the early stages of design there is little concrete information. Assumptions are made regarding the design in order to enable other systems/engineers to the review of 'knock on' effects to the later design stages and enable systems/engineers to participate with few hard

facts. In a knowledge based concurrent engineering environment it is important to manage the distinction between assumptions and recorded facts and account for this distinction in the reasoning process.

Explanation

'Explanation' is the field of research where the computer has the ability to explain and justify its results [SyCara 89]. Attaining user acceptance of a system that supports concurrent engineering design is complex. Agents may reason in a fashion far removed from methods born by the engineer. Computers are likely to come up with a reasoned solution in a short time, whereas if the human has not applied the same reasoning it may not be immediately apparent why the computer has developed a particular result.

Trade-off between alternative engineering ideals

Neither party in engineering design can achieve their goals unilaterally, and therefore cooperation is required in order to reach a solution [Anson, Jelassi 89]. It is important to ensure that the different design perspectives are reviewed and appropriate design trade-offs carefully balanced in order to develop a solution acceptable from the global perspective.

It is important that a concurrent engineering system that provides advice from a life-cycle perspective can trade effectively in the different engineering ideals. In order to trade 'ideals' however they need to be brought down to the same base unit or cost. In many circumstances one relates things to monetary value as the base currency for trading the different alternatives. This is mainly because one is used to dealing in this unit and can more easily judge the extent of the costs involved, and secondly because some units under consideration are likely to be in a monetary unit anyway and therefore save ourselves an arbitrary conversion. However problems exist relating things to cost, for example, what is the cost of a human life?

An example objective in the pumping domain may be to *'meet the head and duty requirements'*. This is a qualitative factor and is either achieved or not achieved. It is likely however that one requires a pump that is 'efficient', and this can be viewed as a general objective, in this case to achieve as *'high an efficiency as possible'*. These general objectives are not usually achieved in totality but show a direction in which a design should take (i.e., go for the more efficient pumps). In design there are many objectives to satisfy (e.g. cost, maintainability, safety) and achieving one objective usually compromises another, and therefore one has to seek a sensible compromise.

Share a 'common' view on the design

For disparate software systems to communicate and share design data a common data repository is needed. The collective design solution has to be comprehensive in the sense that it has to maintain the knowledge pertaining to all design viewpoints and be able to

distinguish between facts and assumptions.

Formulating the problem requires a knowledge representation strategy that can represent:

i. the issues, goals, evaluation criteria, and design constraints [Lander. Lesser, Connell 89]
ii. alternatives under consideration
iii. justifications for design decisions [Sycara 89]
iv. the software systems in conflict
v. the issues over which there is agreement/disagreement [Sycara 89]

3.4. Tools and technologies to support CE

To aid the engineering designers there has been the development of a formidable array of engineering design tools such as CAD, Finite Element Analysis Tools. Simulators, and knowledge based design tools. These tools provide benefits to the engineer such as those purported by concurrent engineering such as for example the rapid checking of preliminary design concepts, sharing the latest design information, and identifying downstream constraints. These technologies that support the CE ideals are diverse, and a review of the principal technologies are covered here. The section has been split into technologies that predominantly manage design information, from those that process design information to support the design process itself.

3.4.1 Managing engineering information

Engineering data management (EDM) in its broadest sense can be defined as the systematic planning, management and control of all engineering data required to adequately document a product from its inception, development, test and manufacture, through to its ultimate demise [McIntosh 92]. There is much data generated throughout a project, and this data is related in many ways. If data is sequentially dependent then this relation must be recorded, e.g. that pump head was derived from height of the inlet vessel. Due to projects being evolutionary, data is neither static, nor fixed in extent. It is important to record the status (level of confirmation) of a data item and its quality (confidence). Motard [Motard 89] refers to these types of system as EIS's - Engineering Information Systems.

The justification for such engineering systems is that engineers spend much of their time on routine tasks such as information retrieval - estimates range from 15% to 40% of the engineers time [McIntosh 92] - and that designers spend between 50% to 80% of their time moving and organising data between applications [Motard 89]. The moral to be learnt is that it is more cost effective to re-use data than to re-generate it.

EDM systems store and accumulate vast amounts of information from many different

sources. They therefore have to provide the ability to store data in a distributed fashion over a computer network and avoid corruption during updates [McIntosh 92]. Several authors [Motard 89] have identified distribution as important in overcoming the performance penalty likely in one huge collection.

Benefits cited by McIntosh cover:

- reduction of lead times, from serial to concurrent environment
- improved quality of products and services
- reduction in development costs during design and development phases
- better data re-use with fewer design errors
- more effective use of engineering time

3.4.2 Tools & Techniques to support the design process

Constraint Satisfaction

A constraint restricts the values that may be assumed by a group of one or more parameters [Bowen, Bahler 93b]. A constraint network is a set of constraints, which are interconnected by virtue of sharing parameters. A major attraction of constraint networks is that constraints support multi-directional inference: information can flow in any direction through a network. Thus for example the impact of a design decision on the options available to the test engineer can be determined by propagating the design decision and its consequences throughout the network.

Design tasks can be viewed as constraint satisfaction problems in which a design must be created within fixed limits on time, cost and materials, and within the bounds of technology. The number of possible interpretations that can be assigned to individual components of a design is large, and the number of combinations of components is enormous. A closer analysis however will often reveal that many of the combinations cannot actually occur [Rich, Knight 91]. By viewing a design problem as one of constraint satisfaction, it is often possible to reduce substantially the amount of search that is required as compared with a method that attempts to form partial solutions directly by choosing specific values for components of the eventual solution. In concurrent engineering the desire to address a broad range of life-cycle issues in a non-sequential fashion during the early design stages introduces two main difficulties [Krishnan, et.al. 90]:

i. the multitude of constraints obscure the designers' understanding of the design by increasing the dimensionality of the problem.

and

ii. the dimensionality of the problem increases the complexity of finding a design solution. It is not clear to the designer which constraints are important and which ones are irrelevant. There is a need to sift the model and identify :

- the active constraints that desire where the solution lies
- the critical constraints that determine the solution and preserve the integrity of the model and
- the irrelevant constraints that can be deleted from the model.

From a concurrent engineering perspective the attraction of constraint networks is that they enable the assessment of the impact of a decision made concerning one phase of the products life-cycle on the later design phases.

Automatic plant layout

The layout of a plant in 3D space is important in the costs of operating the plant, as well as safety and operational concerns. Significant cost advantages are achieved through process synthesis, which optimises the process for heat and mass balance. For example, if one product needs cooling, while another needs heating, then the layout should account for the requirements by placing in the two product streams in an optimal fashion to enable heat transfer to occur. Additional constraints are imposed on layout to ensure that products do not exist in the same plant zone if potential hazardous problems could occur. There are a considerable number of concerns regarding plant layout and it is a very knowledge intensive process requiring considerable experience and intuition.

Madden et al. [Madden, Pulford, Shadbolt 90] discuss a method for generating a plant design purely from conventional flowsheet and process data. The method can also answer questions such as can we build it? Should we build it? and can we afford it? Automatic layout provides an instant view to the process engineers (who are not experts on layout) and can be performed consistent with company practice. The tool provides a fast 'what-if' approach and can therefore reduce any unknown effects of layout changes.

Intelligent computer aided engineering

CAD (Computer Aided Design) has led to significant improvements in the design process through higher quality drawings, and faster turnaround times. Forbus identified a new philosophy or vision for CAD:

> *'It is clear that what is needed if the computer is to be of greater use in the creative process, is a more intimate and continuous interchange between man and machine. This interchange must be of such a nature that all forms of thought that are congenial to man, whether verbal, symbolic, numerical, or even graphical are also*

understood by the machine and are acted upon by the machine in ways that are appropriate to man's purpose.'

Extensions to CAD to support design have been termed 'ICAD' - or intelligent CAD. MacCallum [MacCallum 90] argued that intelligent CAD will only be successful if the following tenets were accepted:

i. design is considered to be an intellectual knowledge-based process

ii. systems do not need to replicate human intelligence; they are required to exhibit behaviour regarded as intelligent

iii. necessary components of such systems are knowledge rich models of designs, the capacity for tacit knowledge, and the ability to learn.

MacCallum discusses the idea of an IDA, 'Intelligent Design Assistant'. The aim of an IDA is to take an active role in the design process and is able to contribute to the work of the designer. The IDA understands goals, processes solutions and assesses situations.

Forbus states the aim of ICAE is to construct computer programs that capture and use a significant fraction of the knowledge, both formal and tacit, of engineers. He visualises a system that; offers basic concepts; warns against unsuitable alternatives; suggests possible solutions; reminds; qualitatively simulates; and diagnoses and refines. Kimura's [Kimura 89] requirements for intelligent CAD are more extensive and cover:

- maintenance/creation of multiple manifestations
- sketching
- transition from conceptual to detail design
- design evaluation
- suggestions of appropriate solutions

etc.

In summary ICAD plays the part of a design assistant: a system that is able to take a pro-active part in designing and is able to bring to bear its particular strengths in design problem solving [MacCullum 90]. The system is closer to a complementary relationship with the designer, but relies on the designer to identify and present the problem, to specify the environment, and to assume responsibility for decisions.

3.5. Design issues in the development of CE support tools

In order to achieve the software advances that are required to realise a true concurrent engineering support environment there need to be advances in the technology to support it. These improvements cover many areas such as communications, data management and

user interface design. These advances in technology will aid in the re-use of engineering data, reduce errors, ensure more effective use of engineering time and reduce lead times. This section covers some of the research issues to be resolved.

3.5.1 Large scale support tools

Resolving conflict

When developing knowledge based systems the option is available to integrate or define knowledge so that in normal use conflict does not exist. In other words, conflict resolution is performed at development time. Obtaining agreement however between a large number of experts on how design should proceed would be an enormous task and would be unlikely to succeed. Maintenance of such a system would also present considerable problems as the addition of new knowledge is likely to affect many areas of the design process where potential conflict would have to be identified and resolved. The alternative is to resolve conflict between agents in operation, this is termed 'run-time' conflict resolution [Polat, et.al. 93].

Constraint satisfaction

In the design process an engineer has to access a large amount of references concerning standards and design guides. A problem with reference approaches is that the designer has to manage a high volume and variety of information. Constraint networks provide a powerful method for maintaining relationships between a variety of different concerns. The networks however suffer from the problem that all the variables are related to each other in some way. In a large single knowledge based system the approach is difficult to implement because the computation can be crippling. It may be considered that a logical solution to this problem would be to distribute the processing across a number of machines. More problems are introduced however as these systems may have their own internal variables that may be inconsistent across systems.

Algorithms

Due to the fact that engineering design environments utilise many different design tools, and that there is no single algorithm that is capable of resolving all parts of the design process, it is unreasonable to expect a single system to resolve the many different design problems. An approach where different techniques, algorithms, representations, and reasoning mechanisms can be applied is required.

Processing ability

Controlling knowledge based systems has become increasingly difficult as applications increase in size [Goldstein 94]. Traditional knowledge based approaches employing global memories fall short of addressing real world problems; as problem size increases, the same

lessons learned by programming languages decades ago are now being re-examined. The combinatorial explosion of rule interactions demands that sophisticated knowledge based applications employ sophisticated architectures and control methods.

The traditional knowledge based approach of developing a single, consistent knowledge base to provide the life cycle perspective poses significant problems. Although designing a knowledge based system with a single knowledge base is certainly easier than integrating multiple knowledge bases [Steer, et.al. 93], the problems posed for knowledge elicitation, maintenance, and the addition of new knowledge are formidable. It has been stated that "it may be much easier to coordinate the activities of 20 medium sized machines than to build a single machine that is 20 times as large" [Uma, et.al. 93]. The combinatorial explosion of knowledge interactions pervades the development of knowledge based systems [Goldstein 94].

It is considered that a distribution of processing not only improves the performance of large knowledge based systems, but also encourages re-use, and many simple systems can be combined and integrated to build large complex systems [Goldstein 94]. Mandiau and Millot [Mandiau, Millot 92] suggested that a system may be so complicated and contain so much knowledge that it is better to break it down into different cooperative entities in order to obtain more efficiency (i.e. modularity, flexibility, and a quicker response time).

3.5.2 Distributed support tools

Communication

Due to the distinction between the various disciplines traditionally involved throughout the design process, and the complexity of these individual tasks, software has evolved with little regard to how to share results or utilise information available in other parts of an organisation. These individual developments are known as 'islands of automation'.

These 'islands of automation' collect pools of digital data that are either generated by software or entered by an engineer. The entering of data is prone to errors and therefore it is important to reduce data entry where possible by sharing data between applications.

The development of central engineering databases where applications can request and store data will overcome the problems of data sharing. However, the amount of information generated during the lifetime of a project is considerable and many authors have suggested the need for multiple databases to overcome the performance penalty in one huge collection [Motard, 89]. With such a wide variety of independent software developments and the need for multiple information systems to communicate, the importance of a standard communication protocol cannot be underestimated.

A STEP protocol (the standard for the exchange of product model data) has been under development by ISO (International Standards Organisation) since 1984 and is a IEDI

standard (technical electronic data interchange). STEP essentially represents the total information content of a product by means of a product information model. The product information model identifies the information required for a product from its early conception to its final demise. The adoption of this standard will enable different software developments to request information from other data sources and pass its results through to other applications.

Another standard has evolved for paperless trading between separate companies. This standard is known as EDI (Electronic Data Interchange) and is becoming increasingly popular in several manufacturing sectors from large industrial companies to the retail and insurance sectors [Knight, 91]. It is essentially a method for suppliers to bill customers, and customers paying suppliers by electronic means.

DAI

In order to avoid the problems with traditional knowledge based systems, an approach to partitioning expert systems into loosely coupled systems is required [Uma, et.al. 93]. Individual knowledge based systems operating in such a distributed environment have been termed *Agents*. DAI (Distributed Artificial Intelligence) is a research area where knowledge is recorded and processed in discrete collections (i.e., distributed), usually referred to as individual reasoning Agents. Having single autonomous agents however poses the need for coordinating them towards solving complex design problems and therefore planning is required. Reasonable models of distributed expertise will dictate that agents hold incomplete local views of the problem [Davis, Smith 88]. Agents may possess knowledge unknown to other agents, maintain different beliefs and evaluation criteria, have incompatible knowledge representations, or may just be logically inconsistent with each other [Bond, Gasser 88]. Conflict is therefore inherent where expertise is distributed.

Besides the problem of maintaining rules in distributed knowledge bases, the ability for agents to learn, and therefore improve their performance is also a problem. Learning is important where there is iteration in the design which is a particular facet in a concurrent engineering environment. Learning will enable an agent to determine when particular proposals fail for some specific design criteria which it can identify. Other agents however may change their knowledge, viewpoint or goals, and therefore present problems in reflecting these changes in the knowledge which other agents had learned. Agents could suffer the same problems as humans do, "we do this because we have always done it this way" (i.e., knowledge has been learnt from a pattern rather than rational engineering judgement).

Conflict

Reasonable models of distributed expertise will dictate that agents hold incomplete local views of the problem [Davis, Smith 88], and maintain different priorities and design goals. These different concerns are the inherent cause of conflict that has to be resolved.

Conflict is a disagreement between two or more viewpoints on some value proposed in the design. These conflicts arise from the differing needs of the separate disciplines and knowledge based systems. These needs comprise both *goals* and *values*. Goals identify particular directions that a system will look for a solution -for example - produce a design for heat transfer system. The differing values identify the importance with which an expert or knowledge based system regards a particular issue (e.g. safety against cost).

As has been previously identified, it is unreasonable to expect design agents to all utilise the same reasoning strategies and one can therefore expect their knowledge representation strategies to differ. Differing knowledge representation strategies will prevent agents from communicating and sharing information in order to resolve conflict. Standard methods for identifying and resolving conflict are therefore required.

Assumptions

In the early design phase very little information is available. This presents the problem of how to ensure a life-cycle perspective is achieved when most of the design agents can not participate due to a lack of information. This problem can be alleviated through the use of assumptions and constraints. Without information, the agent can assume a value of a design attribute that may be the most credible. This enables detailed design to proceed on a lack of detailed design information, which is obviously lacking in the early phases of a design project. Minsky neatly described the importance of assumptions in normal day life:

> *"isn't it remarkable that words can portray individuals ? You might consider this impossible considering all the things that there are to say. It is because we all agree on so many things that are left unsaid. When reading a book, we assume that all the characters are possessed of 'commonsense knowledge', and we agree on many generalities which we call 'human nature'. e.g. hostility evokes defensiveness.....Default assumptions embody some of our most valuable kinds of commonsense knowledge: knowing what is usual or typical." [Minsky 86]*

Assumptions, due to their nature, are obviously the first area for analysis when they lie on a conflict path. Identifying assumptions however may be problematic.

Assumptions are usually formed from some normal life experience on what is 'the usual' or 'standard' approach. In the case of a requirement to pump fluid, with no further information it would be ok to proceed on the basis of the pump being of a centrifugal type as most pumps are. Given a little more information, say for example the pump has to pump fluid from the bottom of a mountain, then a positive displacement pump type may be assumed. In a distributed environment, if problems are resolved, as and when decisions are made, one could potentially have problems with regard to the initial assumptions being inconsistent. In a perfect world, it would be sensible to assume that each person or agent would assume values for a particular design attribute that were consistent (or the most common). In other words you could imagine as each agent starting from some kind of

basic design premise. If one starts from this perfect world premise where all initial assumptions are consistent in the global sense, and that any changes to the assumptions (from some reasoning process) is communicated between agents and any conflict resolved, then the result will be consistent. If however inconsistent assumptions are made at an early stage of the design process, and these are not properly communicated throughout the normal design process, then these will not be recognised until final system integration.

An important point to note that if the assumptions are initially inconsistent, it does not matter how much consistent communication takes place, the result can possibly be inconsistent. This problem can be avoided by automatically reviewing the assumptions on startup although the overhead can potentially be extremely high, or by not allowing assumptions to be made unless through the normal design communication process.

Experienced design teams produce more accurate designs more quickly and with less conflict. This could be due to the reason that they are more likely to make the correct assumptions to start with that are a result of reviewing systems on which they have all worked together.

3.5.3 Representations

With regard to a CE support system, one has to consider two different representation systems: how the evolving design is represented, and how agents store their knowledge. If a standard representation is used for agents, it is possible to unify the knowledge before processing to highlight conflicts, although this presents a few computational problems, some that are potentially unbounded.

Engineering designers each have their own conceptual systems and different terminologies. Sharing this information with other designers can be problematic and cause conflicts through misunderstandings. On a chemical site for example, 'process water' can mean different things to different people on different plants. Shaw and Gaines found that:

> *"One problem of eliciting knowledge from several experts is that experts may share only parts of their terminologies and conceptual systems. Experts may use the same term for different concepts, use different terms for the same concept, use the same term for the same concept, or use different terms and have different concepts. Moreover, clients who use an expert system have even less likelihood of sharing terms and concepts with the experts that produced it." [Shaw, Gaines 1989]*

We can deduce from this that a common vocabulary is required to ensure communication is consistent and non-ambiguous.

Agents that maintain a common representation - such as distributed truth maintenance systems - will be able to communicate and share information. It has been identified that

it is unreasonable to expect design agents to each utilise the same reasoning strategies and one can therefore expect their knowledge representation strategies to differ. The knowledge that agents will possess will also differ, any reasonable models of distribution require incomplete local views of the problem [Davis, Smith 88]. Agents are likely to hold inconsistent beliefs to one another if conflict resolution is performed at runtime, as opposed to development time. These conflicts, and the conflicts that can be expected in translating possibly incompatible knowledge representations will all present areas of conflict that will have to be resolved.

A major failing in most intelligent systems is that they are unable to undertake an open interactive dialogue about the thing they are designing [MacCullam 90]. In order to understand a problem you need to have a working model of it. A method for modelling a problem domain, constructing a 'common view', and ensuring that different agents can communicate (may be through a translator - or 'wrapper') is therefore required.

Maintenance of a 'common view' also provides difficulties. Modifications to the view can impact on the developments already made, and constructing this view in the initial stage will prove complex.

3.5.4 General data management approach

The management, structure and storage of data has posed particular problems for the process industry due to the vast amounts of information generated throughout the lifetime of a project and from such a large variety of sources. It has been highlighted that a single database system would be too large and unwieldy for a single computer.

The role of Engineering Data Management (EDM) systems is defined as the systematic planning, management and control of all engineering data that is generated. There are many problems that an EDM has to address;

a/ Maintaining consistency of information

b/ Identifying the relationships between data

c/ How data is organised

d/ Enabling different views on the data

Maintaining the consistency and integrity of information is a big problem in the engineering industry. The appropriate people must be informed of changes to a design and the appropriate re-work performed. Any problems in communication or failure to identify the parties involved can lead to project delays, production problems or even hazardous designs.

Due to the variety of experts who require and analyse data there are many different views on information in the database. Each discipline requires a view that is relevant to the type of work at hand. The engineer does not wish to sort though masses of information to identify which data is relevant. The structure of information and the design of consistent and friendly user interfaces is imperative to the success of an engineering support system.

The process industry also requires flexibility in the type of data stored. Information such as matrices, arrays, text, graphical drawings and various dynamic structures are required to be recorded. The variety of data that is stored and the relationships between these items of data makes the task of maintaining consistent databases extremely difficult.

The relational database architecture is a popular and powerful method for the storage and retrieval of information. Information in such databases requires to be 'normalised' and the relationships between data items clearly identified.

The adoption of relational architectures in engineering data management systems however has not been successful. There have been many criticisms of the performance of such systems and there are many ragged relations and a lot of redundancy.

Object oriented databases are becoming a popular research area due to the ability for components to be stored and modelled as objects to any degree of complexity. Object oriented designs enable objects with intricate and dynamic relationships to be specified and modified with little or no knock on effects to other areas of a design. Object oriented programming languages are also becoming more popular which enable objects to be manipulated and stored. The object oriented programming style enables object code to be developed independently of other objects. This separation of object development and the 'real life' association of objects enable the object code to be re-used in other applications which reduces the development times. Object oriented styles however are not without their problems. The primitive relationships that define objects are unable to model many of the requirements of an engineering discipline.

3.5.5 User interface engineering

However advanced or complete an engineering system may be it is worthless unless it can be used by the engineers. The development of user interface standards and 'Windows' interfaces enable the production of user friendly and consistent user interfaces.

The engineering community requires the use of many varieties of software packages and therefore the maintenance of standards to enable engineers to move around this software is important. Due to the complexity and the masses of information required in a project, the hardware is likely to have a complicated structure and information resides on several computer systems. The engineer should be shielded from the underlying complexities of the communication and data storage systems and should view the system as a single entity.

The development of user interfaces can be further enhanced with knowledge based facilities that learn from the engineer. Common actions and sequences of commands that are performed by an engineer are recorded and analysed. These actions and commands can then be played back automatically by the system when particular positions are recognised. These features will further enhance the useability of the software and improve the performance of the engineer.

3.6. Summary

Concurrent engineering has been proposed as a way forward to improving productivity through the use of multi-disciplined teams and parallel working practices. The complexities of engineering design and the many interrelated design issues that exist between the various design disciplines however evade rigorous human analysis. The integration of engineering design tools is beneficial in a concurrent engineering environment where engineers can freely exchange information and improve design turnaround.

The requirements of a framework to support the design process have been identified in this chapter. These features have been described from the viewpoint of the design process itself, and in terms of the technology to implement the features:

Generic features of an engineering design tool:
Impact assessment
What-if analysis
Design support - not automation
Life cycle review
Participation from other engineers

Technology features of an engineering design support tool:
Distributed architecture
Reasonable on-line response
Incremental development
Knowledge based
Ability to integrate with other systems
Cooperation with other systems - resolving 'conflict'
Manage and maintain design consistency
Handle uncertain information
Explanation
Trade-off between alternative engineering ideals
Share a 'common' view on a design

In recent years computing technologies have evolved throughout many disciplines and improved the productivity and performance of design groups. This chapter reviews some

of the main technologies that support CE from those that manage information (EDM, EIS) to those technologies that can apply in the engineering design process. The industry however has still much room for improvement. Much of the computing technologies have evolved within the confines of single departments which has lead to the inability of computers to communicate and share information. This has lead to problems maintaining the consistency of data and the misuse of engineering time trough having to re-input data. The development of standard interfaces and data exchange standards has aided the integration of engineering design tools. Integration of knowledge based systems involved in various stages of the design process however poses significant problems. The development of a single large knowledge base to encapsulate life cycle concerns poses problems in development, maintenance, management, and performance. Partitioning of a knowledge based system to encapsulate life cycle concerns appears the best way forward, although there is now a need to coordinate a set of autonomous agents towards solving a complex design problem. This chapter has covered these issues that relate to the development of tools to support the design process.

Knowledge based systems for concurrent engineering must provide advice and solutions that resolve the life-cycle issues, and thereby present solutions that are a result of the collective expertise available to the knowledge based systems. This therefore requires knowledge based systems to share a standard protocol and data representation, as well as identify and resolve design conflict. The chapter covers in detail the features and requirements of a tool to support the integration of design and knowledge based tools into the design process.

A computational support environment will aid in reducing the data management overhead, reducing transcription error, improving design decisions through the provision of knowledge based expert advice, enable the engineer to spend more time on 'real' engineering problems. The next chapter will describe the theoretical design of the CDEX framework that accounts for the requirements identified in this chapter.

Chapter 4. Framework to support Concurrent Engineering

4.1. Overview

In this chapter the conceptual framework is explored that can support the requirements of concurrent engineering design laid out in the previous chapters. The conceptual framework lacks the necessary detail required in order to develop a framework in code, but provides an understanding of the different components required in the framework and an understanding of their function. The next chapter - CDEX - will introduce the physical implementation model developed in the PhD that realises the conceptual framework described in this chapter.

This chapter will present an example 'model of use'. This model of use is a depiction of how an engineer may use the CDEX system from our understanding of the requirements previously identified. The engineering design tasks need to be formalised in order to develop a framework that manages the design process. This chapter formalises the design approach by classifying the different types of engineering design. The chapter then progresses on a discussion of framework technologies and framework structures. The framework is presented as a layered model, with each layer supporting a different aspect of a system required for supporting design such as data storage, negotiation, and communication. The ability to represent engineering 'value' is important in the negotiation process and this chapter has described the formal approach to representing 'value' using utility theory and objective hierarchies.

4.2. Model of Use

In order to satisfy the requirements of a concurrent engineering support tool, a 'model of use' has been constructed highlighting the style of interaction between the engineering user and the design framework.

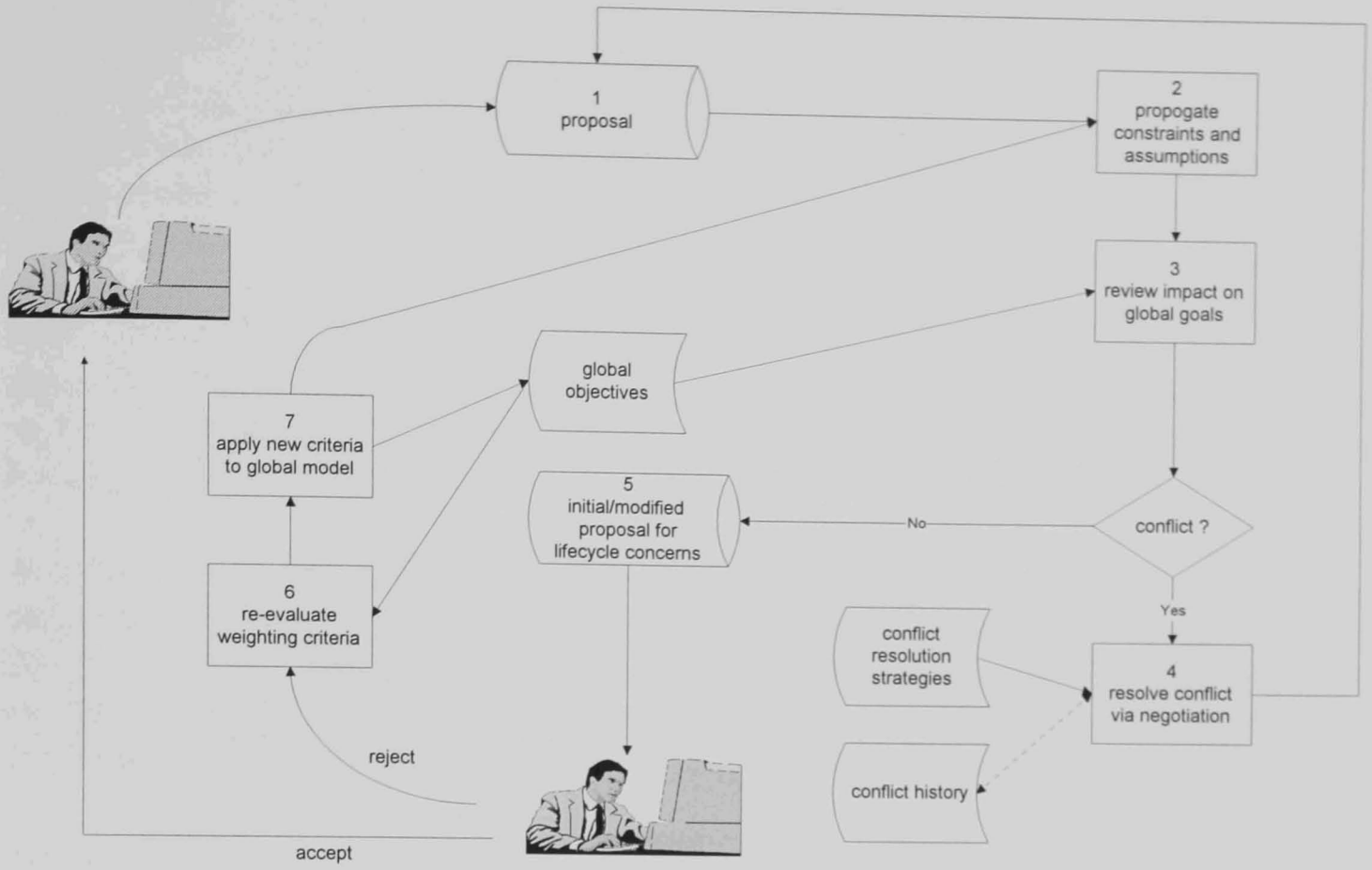


Figure 7 Model of use

Initially the engineer specifies the requirements and constraints that he has identified from the basis requirements. He is supported in his task utilising familiar design tools (e.g. flowsheeting software, CAD, sizing programs) and knowledge based systems.

The engineers specification (1 - *see 'Proposal' in Figure 7*) is cast into the design space for review by other interested agents. The detail in the early design phases is limited and is not usually in a form that can be easily interpreted by other agents whose perspective is predominantly concerned with issues in later design stages. In order for a proposal to be reviewed by other agents, the design data and constraints are propagated throughout the other design agents (2 in Figure 7). During this process agents may make assumptions concerning information either from the design constraints or from the factual design data specified. In this way agents that do not usually participate in the early design stages due to a lack of design information can identify their concerns around the direction the design is taking, thereby accessing the potential impact of a design decision.

An agent may determine that a design proposal is unacceptable due to the fact that a constraint has been violated. This would be taken to be a hard constraint, i.e. one that can not be compromised. In the selection of engineering equipment, such a law may be the basic requirement to ensure the temperature in a system does not exceed a critical limit. Agents at this stage are not in a position to reject a proposal due to the fact that they believe the design is not in their favour (a soft constraint). An evaluation of the design

must be considered from a global perspective. In concurrent engineering, design proceeds in a cooperative manner where the emphasis is on shared goals and team ownership of the end product. It may be the case that although a proposal is not close to an ideal solution, it may solve a host of problems that when taken from a global perspective will seem an adequate solution. For example, it would not be unusual to find a pump on a process plant that is more expensive and less efficient than that required. The justification for such a decision however may be twofold. Firstly, the engineers can standardise on one pumping unit, therefore reducing the number of spare parts in stock. Secondly, a non optimal pump may be purchased so that only one spare pump is needed for backup (i.e. one backup is shared between many different pumps). This cost saving may not be immediately apparent in the early stages of design where the more expensive and less efficient pump design may have been quickly discounted.

A design is reviewed from the global perspective (3 in Figure 7). Agents are allowed to raise conflicts and evaluations (highlighting their preference) concerning the direction the design is taking. This provides additional information to the negotiation mechanism in deciding a suitable way forward. This more specific knowledge will also help avoid conflicts that an agent may recognise to cause potential problems further in the design.

The requirements on the negotiation mechanism are to firstly identify the primary cause of the conflict, and then to select the most appropriate strategy to adopt in resolving it (4 in Figure 7). The root cause of the conflict may not be where the conflict was identified. The conflict may be due to some assumption made early in the design that has to be identified and corrected. After the conflict has been identified, the situation has to be analysed and the most appropriate conflict resolution strategy applied to resolve the problem. This may be a compromise, search for other alternatives, integrative, or in the worst case, a blind search for an acceptable solution.

The results from the negotiation process will be a solution that is acceptable from the global perspective. It is possible for the negotiation process to fail in finding a solution to the problem. This may be due to agents holding contradictory knowledge (more than one competing hard constraint) that was either incorrectly interpreted at the knowledge elicitation stage, or incorrectly entered into the knowledge base. In this case the engineer will have to be informed and the conflict left to human deliberation. In the current human situation, the engineers can discuss the misunderstanding and essentially update their viewpoint in 'real-time', updating and exchanging information until their viewpoint is consistent. Unless the computers share the same knowledge representation, or there is a set of comprehensive wrappers [Londono, et.al. 89] (translation mechanisms), the agents will not be able to share knowledge. Due to the different design representations that will be utilised by the different design mechanisms, and the problem of determining the quality (correct) knowledge provided by the conflicting parties, the problem of knowledge sharing has not been tackled in this framework.

If the engineer rejects the solution then he will be required to state why it is inappropriate

(6 in Figure 7). This reason will be stated in terms of the global objectives, for example, the 'proposal is too expensive', or the 'solution does not satisfy the safety requirement'. It is important to enable the engineer to override the global concerns as he is likely to have experience affecting the design that is not known by the knowledge based agents. It also enables the engineer to remain in control throughout the design. A single objective hierarchy can never be formally verified due to the subjective element that is likely to exist between engineers'. Enabling the engineer to remain in control will remedy the type of problems that can be caused by missing information and subjective elements, while providing the engineer with life-cycle information that they might not necessarily have been aware of.

When the engineer has provided a reason for the design not satisfying his requirement, the new weighting criteria will be used to update the global model (7 in Figure 7) so that the same problems do not re-occur. The proposed solution is then modified to account for the engineers' concerns and then proceeds through the same process of propagating the effects and reviewing the impact on global objectives.

4.3. Framework technologies

4.3.1 Rationale for a framework approach

As stated previously, an environment to support concurrent engineering must support a diverse range of computational methods. There does not exist a single computational method that can resolve the many different design problems. A method of integrating the disparate design systems and methods would be to create interfaces between the different packages thereby enabling communication. The problem with this approach is the number of interfaces that would have to be developed between the different systems. These interfaces would be required to be more complex than just the normal 'run of the mill' style translators as the different packages would have to be able to resolve conflict between themselves, thereby complicating the interface. A framework on the other hand provides a single interface with which all packages should conform. The framework will provide the necessary capabilities to enable the different packages to work together in resolving the design problem.

Talukdar and Fenves [Talukdar, Fenves 89] have outlined the following needs of a framework to support design:

- provide formal ways of stating the problems of concurrent design
- provide visualisation aids to help devise strategies for solving problems
- provide implementation aids to help translate strategies into working systems

4.3.2 Framework strategies

Talukdar has stated the problem of concurrent design quite simply as *"how can the propagation of deleterious effects from one task to another be reduced to tolerable levels ?"*. Framework strategies for resolving the problem have fallen into two broad categories: 'preventive or lookahead strategies' and 'corrective or feedback strategies' [Talukdar, Fenves 89]. Preventive strategies attempt to avoid conflicts before they occur. This can be achieved computationally through techniques such as constraint satisfaction, reasoning through assumptions, and general inference mechanisms. Corrective strategies attempt to resolve conflict when it occurs, and this is achieved through backtracking mechanisms.

A concurrent engineering team principally attempts to develop a preventive strategy. By enabling the many engineering groups to voice their opinions at each design stage, they are anticipating the conflicts before they occur and therefore progressing the design to avoid them. Corrective or feedback strategies are expensive in engineering design as design modifications may have considerable knock on effects to the later design stages. This is especially so in the early design stages where the decisions are generally larger grained.

Due to the nature of a framework to coordinate and enable communication between potentially diverse information sources, many issues are being tackled which are in the area of distributed artificial intelligence - or DAI. How to recognise and reconcile disparate viewpoints and conflicting intentions among a collection of agents or knowledge sources is one of the basic problems in DAI [Bond, Gasser 88]. DAI has divided the world into two arenas, Distributed Problem Solving (or DPS), and Multiagent Systems. DPS considers how the work of solving a particular problem can be divided among a number of modules, or 'nodes' that cooperate at a level of dividing and sharing knowledge about the problem and developing solution. Multiagent systems on the other hand deal with coordinating intelligent behaviour among a collection (possibly pre-existing) autonomous intelligent agents, how they coordinate their knowledge, goals, skills and plans jointly to take action or to solve problems. In Multiagent systems the task of coordination can be quite difficult, for there may be situations where there is no possibility for global control, globally consistent knowledge, globally shared goals or global success criteria, or even a global representation of a system.

4.3.3 Agent structures

The benefits of an agent approach to design are typically those proposed in the object oriented design paradigm. The agent approach would require the systematic reduction of the problem into different design spaces, therefore providing spaces that are easier to define, maintain and easier to comprehend than that of the system as a whole. The decomposition of the design problem into different design spaces may also identify spaces that are generally applicable in other design problems, therefore re-use is a possibility. Klein and Lu [Klein, Lu 89] are supporters of the agent problem solving approach for a

number of reasons:

i. improved comprehensibility

Runtime conflict resolution enables the maintenance of different bodies of expertise. These bodies of expertise can be produced by individual human domain experts without resolving the inherent knock on effects in other design areas. If all conflicts are resolved at development time then it would be difficult for an expert to maintain and modify the knowledge base without consultation with all other design diciplines.

ii. increased extensibility

The ability to add new bodies of expertise without the need to maintain consistency with all other knowledge sources. Runtime conflict resolution helps insure independence of the bodies of expertise in a complex KB system. The principles of modular design and implementation tell us that by structuring a complex problem into self contained modules the system is easier to build, debug, and maintain and is less prone to errors [Uma, et.al. 93].

iii. increased flexibility

When conflicts are resolved at run time instead of development time, there is flexibility in the choice of conflict resolution strategy adopted.

iv. involving human problem solvers

One of the more compelling reasons for using runtime conflict resolution strategies concerns the role of humans in cooperative design systems. The use of runtime conflict resolution strategies constitutes a better model of how cooperative design in human design teams takes place and thus provide a more natural framework for systems with human and automated participants.

The fundamental advantage of runtime conflict resolution, then, is that it constitutes a more realistic model of cooperative problem solving than does development time conflict resolution, both by constituting a better model of human group problem solving, as well as by reducing the complexity of the individual design agents to more manageable levels.

The problem arrises however that although the advantages of decomposition are clear, is it really important that systems are distributed, operate in parrallel, and is an important facet in meeting our requirements? One concern is performance [Davis, Smith 88]. For such a potentially large system to be designed and implemented in a way which does not support distribution would prove to complex for a single computer to handle. The different systems required to support the design (different agent capabilities) could potentially

require computational support in the form of simulation, mathematical modelling, finite element analysis, and knowledge based systems for example. A distribution of processing would appear sensible. From the list of potential design systems just covered, one would also expect a variety of different algorithms to be utilised in the design process - not a single 'generic' algorithm that would be applicable for all design problems. Such an algorithm is unlikely to exist [Londono, et.al. 89]. Given a requirement for different algorithms it is easy to imagine the agent approach being a sensible 'split ' between the different technologies just described. One may say that having distributed agents just complicates the problem due to the additional requirement for complex negotiation or conflict resolution mechanisms to resolve the differences between the systems. They may argue that with knowledge based systems, the best approach would be to 'unify' the problem space during the development life cycle, thereby identifying competing constraints, inconsistencies and contradictions. Although this would reduce the need for complex conflict resolution strategies, resolving conflict during development would impose a massive overhead. There is a grave possibility that nothing will ever be agreed. The potential impact of design decisions, and the vast number of related design issues will create a vast number of conflicting situations that would require considerable time to resolve - even if it was possible that reasonable conclusions could be attained. There would also be considerable maintenance issues in a non-agent framework. Any additional knowledge, or modification of knowledge, could require and extensive analysis of many other issues which it effects. As the system grows in size, so does the effort required to define new knowledge. This would impinge on the development of a concurrent engineering framework, and would essentially make the system impracticable.

The idea presented above of a single system that has been 'unified' to avoid the problem of conflict during the run-time design process can be simplified by having a single 'super agent' or 'meta agent' that is responsible for resolving all conflict. This simplifies the conflict resolution process somewhat as there is no 'real' conflict as their is only one party involved. This party however is 'smart' in the sense that he is informed by all the systems involved, rather analogous to a court judge. Whatever decision the super agent makes will be taken as the final judgement. In the CEPS approach by Lander et al. [Lander, Lesser, Connell 89] one of the motivating factors is the need to find a solution when there is no strong global model of correctness or optimality. This occurs when global evaluation criteria are absent, when global evaluation criteria are too expensive to compute, or when a global evaluation is some combination of a set of locally computed evaluations. The last case occurs when the problem is decomposed in such a way that each agent has enough expertise to evaluate some part of the solution, but not all of it. To compute a comprehensive evaluation, a super agent would be required, yet it is not always desirable or even possible to build systems with such an all-encompassing outlook. Considerable effort would be required in order to relate and tradeoff the views of all people involved in order to determine an optimal solution with regard to company policy [Sycara 89]. There would additionally be a large cost in eliciting this global evaluation model, for example, relating pump reliability, cost, corrosion, probability of failure, cost of failure, maintenance, depreciation etc.

It is questionable as to whether seeking optimality in design is a plausible goal. Optimality is also subjective because personnel opinions are involved. An individuals frame of mind at any one time can greatly effect how they perceive a particular solution.

Agents tend to be arranged in complex hierarchies, that is, structures with multiple levels in which agents can report to two or more agents above. Some of the agents perform as operators and some as managers. Managers set goals, decompose tasks, and assign tasks to operators. The purpose of a design system is to select and implement computational paths for performing given design tasks [Talukdar, Fenves 89].

4.3.4 Negotiation and resolving conflict

Negotiation is a particularly undefined problem due to conflicting or inconsistent goals. Design decisions are also tightly coupled where the modification of one design decision may effect previous design decisions, this makes the design process somewhat inefficient and is therefore not amenable to traditional AI techniques [Sycara 89].

The approach taken in CEF [Lander, Lesser, Connell 90] to resolving conflict involved information exchange among participants (agent contraints etc.). CDEX does not enable information exchange in the fashion of CEF due to the requirements imposed on external systems to operate in a specific fashion. CDEX however does implement a minimal approach to information exchange that allows an agent to present problem keywords to describe a conflict, which itself can be classified as information exchange of sorts. CEF justifies its approach through the fact that the human motivational factors are not present in machine agents. CDEX on the other hand attempts to develop an approach using the objective hierarchy and utility theory, whereby these motivational factors are explicitly modelled and used in the negotiation process.

4.4. Engineering design

4.4.1 Engineering Design Classification

In order to support the design process CDEX must be able to request the services of the disparate software systems to perform specific design tasks. The CDEX mechanism relies on the three different types of engineering design classification: 'selection', 'synthesis', and 'parametric' design [Kannapan, Marshek 92].

Design 'synthesis' is the joining together of components that fulfill a particular design function. For example, a heat transfer system for indirect heating is made up from a pump, a vessel, a pipe, heat exchanger and so forth. Design 'selection' is where a higher level description/requirement for a device is specialised to a more specific requirement. For example a heat transfer system may be specialised to an indirect or direct heating method,

a pump may be specialised to a centrifugal or a positive displacement pump type. 'Parametric' design is where values are assigned to the attributes of the component, for example, impeller size for pump, trays in a distillation column, and heat transfer ratio for a heat transfer system. In engineering terms parametric design accounts for what engineers commonly refer to as the 'sizing' process (see Figure 8).

These three types of engineering design are iteratively applied throughout the detailed design process. For example, in a chemical plant design, the process flow diagram is first synthesised - the putting together of process blocks that overall achieve the required process objective. A process may be 'mixing' for example. From here a selection process is performed to determine the method of mixing. There are a number of mixing methods: a normal stirred tank, a special mixer pump, a static in-line mixer, or may be just simple combination of flow streams through a T-bend. The selection process will be responsible for selecting the most appropriate method from analysis of the requirements (reactivity times) and capabilities of the different components to choose from (ability to mix fluids). If a mixer is chosen, then the size of the mixer must be decided (parametric design). The size of the mixer will again depend on the process requirements (volume and type of fluid to mix) and the capabilities of the component selected (guaranteed contact time).

There is an additional design function known as 'innovative design'. Innovative design is required in engineering design where there is not a set of components that can be connected to match a requirement, and therefore a new component must be developed. This is the least used design method in relation to engineering design, and the process industry is wary of adopting novel approaches when tried and tested approaches and components exist. Various AI techniques have been applied in this area covering analogical reasoning and genetic algorithms. CDEX does not concern itself with this method of design.

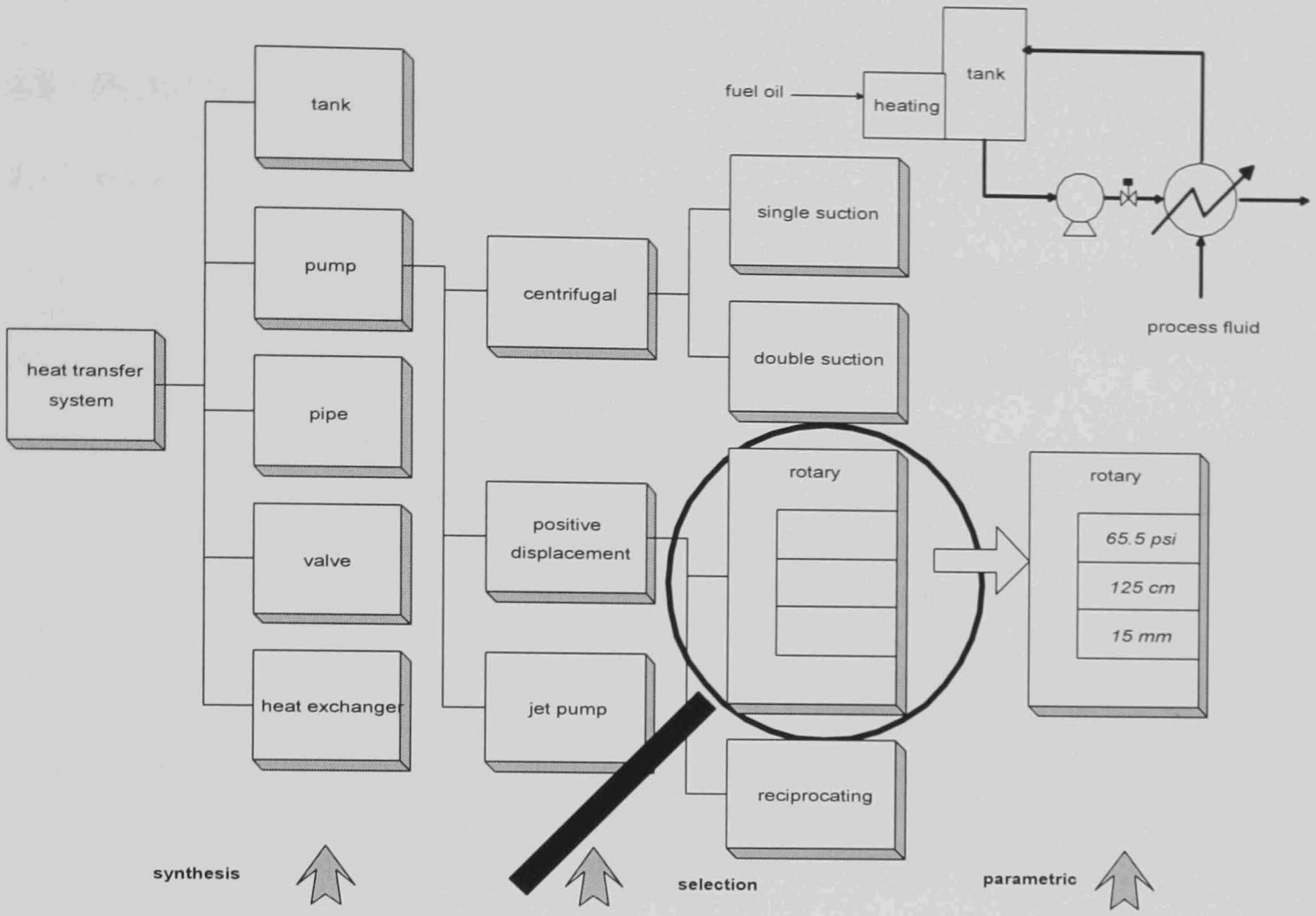


Figure 8 Diagram depicting the three types of engineering design classification.

From looking at the three different types of engineering design presented, it can be seen that there is a clear mapping onto object oriented technology. This mapping is explained later in the chapter.

4.4.2 Design as a top down process

Engineering design is typically a top down refinement process. In the design of a chemical plant, an engineering design team will develop the Process Flow Diagram (PFD) which depicts the major processes in a plant that will produce the required product. The PFD then goes through a refinement process, where each of the processes identified will be decomposed into smaller processes, items of equipment, or plant systems. The process is complete for the process engineering function when the piping and instrumentation diagram is complete (P&ID). The P&ID depicts all the items of equipment (control valves, vessels etc), control instrumentation, utilities, and electricals. From the example depicted in Figure 8, a heat transfer system (HTS) is the initial requirement (which may be derived from the PFD), which has been decomposed into more detailed equipment items

that can be pieced together to fulfill the process intent of the HTS.

4.5. Modelling the needs of the user

4.5.1 Why model a need ?

In the engineering design process each discipline has their own priorities, objectives and goals to satisfy. The same issue applies in a distributed environment, where different agents encapsulate the different experience and priorities of its designer.

In CDEX these priorities are modelled in an objective hierarchy [Keeney, Raiffa 76]. Each agent may have their own methods for determining the utility of a design, but at the same time must be able to relate these objectives to a predefined objective model which everyone can understand. This is important in facilitating the negotiation process as CDEX needs to identify competeing goals, and be able to exchange goals to determine if conflict is over the goals of discussion, or the effects of selecting a particular design option. For example, some higher level objectives that are shared by all the design disciplines may be that the solution be cost effective, meet the duty requirements (rated and alternate) and be operable and maintainable. An example lower level objective may be to ensure 'an adequate safety margin on pump head' which would be an important objective to a process engineer.

4.5.2 Local or global goals ?

In the human workgroup it is difficult to ensure all disciplines in a workgroup share the same value structures as value structures are affected greatly by peoples experiences. For example, someone involved in a serious accident concerning the release of chemicals from a leaking pump, is more likely to put greater emphasis on sealless pumps if the fluid being pumped is carsneogenic (i.e. causes cancer). Value structures can also be affected by current topical problems covered in the national press. In process plant design for example, more emphasis would be placed on 'designing in wingles' if a lack of wingles had recently been the cause of a recently publicised accident.

In a distributed software environment there is a choice to be made with regard to utilising local or global goal structures. The solution can be either to:

a/ Have goals that are local to the agent concerned, combined with a common global goal hierarchy to enable comparisons to be made between agents,
b/ Have goals that are local to the agent concerned which can somehow be combined to determine a global hierarchy or
c/ have no local goals but a global goal hierarchy

Each of these solutions have their benefits and problems. Having local goals poses fewer problems in structuring a global goal hierarchy of values - over which there is likely to be a lot of discussion and argument. Having local goals however requires each person/agent involved to develop and maintain their own local goal hierarchy. Combining local goals to form a global hierarchy - or a centrally agreed global hierarchy - only benefits us in the sense that their is no conflict over values (i.e. agents will always agree on the design approach if they have the same knowledge of the problem). The problem of dealing with conflict cannot be avoided as it is unlikely that agents will share the same set of knowledge (otherwise why are they distributed?). In concurrent engineering it is an ideal to share goals. Engineers should be end product responsible and should therefore be keen for all stages of the life cycle to be performed following the best design practice, and accounting for the life-cycle concerns. Formulating this global hierarchy however is difficult, and the maintenance issues are considerable and vulnerable to changes in the system (i.e. new agents with different priorities).

4.5.3 What is utility ?

A simple but explicit description that adequately sums up utility theory:

> *"Utility is a function that maps a state onto a real number, which describes the associated degree of happiness"* [Russell, Norvig 95]

If one state is preferred to another - then it has a higher utility. Utility is therefore a function that maps a state onto a real number, which describes the associated degree of happiness. A complete specification of the utility function allows rational decisions in two kinds of cases. First, when there are conflicting goals, only some of which can be achieved, the utility function specifies the appropriate trade-off. Second, when there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed up against the importance of the goals.

Difficulties in utility theory

In many circumstances things are related to money as the base currency for trading of different alternatives. Trading in money is something in which most of us are familiar and in most cases are happy with [Gray, Starke 84], and some factors which are neccessary within the analysis are already based on a currency therefore simplifying the process. Sometimes however it is found that it is very difficult to relate values to an underlying currency, for example, how do we account for the cost of a life when trading off safety measures against cost ? The 'cost of a human life' is an interesting case. Many people do not like the idea of quantifying the cost of a human life - it looks heartless. The fact remains however that there is a cost applied to life in many situations but one usually fails to quantify its value. Keeney and Raiffa also note:

> *"...we have a cherished symbolism about the sanctity of a single life...perhaps our morality has gone astray when it comes to numbers. Emotionally we get choked up about a little girl getting killed - especially if we can see her picture - but we do not feel emotionally touched by thousands of people being wiped out by a tidal wave or an earthquake. Somehow we must learn that our grief should rise monotonically with the magnitude of a catastrophe. Numbers are important..."* [Keeney, Raiffa 76]

There are other problems concerning utility theory apart from the difficulty in expressing numerical weightings. These concern [Klein, Lu 89]:

i. If the weightings have been derived from several experts, it cannot be assumed that the experts applied the same semantics when assigning the weights.

   A parable in the bible provides us with an example of variable value when accessing the value of weights. An old lady who gave her last coin to a beggar, gave far more than a rich man giving 1000 coins. Since value is subjective between people it is very difficult to develop an overall concept of value for an item that is applicable across all people.

ii. Changing the behaviour of a system may require changes to a potentially large number of weights.

iii. The rationale behind a decision cannot affectively be described using constraint weightings which make solutions derived using the method more difficult to understand.

Tools for deriving utility

Tools are available to support an engineer in identifying and trading-off the rating of issues [Anson, Jelassi 89]. POLICY-PC, DECISION ANALYSIS SYSTEM and PREFCALC are all tools for performing 'subjective preference elicitation and analysis' . Negotiators first assign preference values to randomly generated contracts. Next, an algorithm derives, for each negotiator, the implied weight (or important weight) for each specific issue or alternative varied in the contracts. Whether explicitly or implicitly derived, the preferences are then available for automated evaluation methods. The ultimate objective for analysis is to identify integrative solutions that maximise the utility of both parties. When many issues or possible solutions exist, such calculations can be extremely difficult without computer assistance.

4.5.4 Objective Hierarchy

In the engineering design process each discipline has their own priorities, objectives and goals to satisfy. The same issue applies in a distributed environment, where different

agents encapsulate the different experience and priorities of its designer.

CDEX models these priorities in an *objective hierarchy* [Keeney. Raiffa 76]. Each agent may have their own methods for determining the utility of a design, but at the same time must be able to relate these objectives to a predefined standard objective model. This is important in facilitating the negotiation process as there is a need to be able to identify competing goals, and be able to exchange goals to determine if conflict is over the goals of discussion, or the effects of selecting a particular design option. An example set of higher level objectives in CDEX that are shared by all the design disciplines are: cost effective; meets duty requirements (rated and alternate); maintainable; operable; and meets the physical constraints.

An example objective hierarchy is shown is Figure 9. The objective hierarchy is a tree structure with the broad objective at the top of the tree, with the most detailed objectives as the leaves of the tree (those objectives without children). An objective is achieved if its child objectives (or sub-objectives) are satisfied. The degree to which an objective is met by its children is not always the same, and therefore a weight is associated with each sub-objective that denotes its contribution to the parent objective. The weights assigned to the sub-objectives must total 1, i.e. if the sub-objectives are completely satisfied then the main objective is fulfilled (i.e. equal to 1). These weightings represent the tradeoffs that can be made between the various objectives and provide the indication of how important particular objectives are - in this case to particular agents. The lower level objectives should be at a level of detail whereby an attribute can be measured in the domain which is a measure of the objective itself. For example, the cost of spare parts for a pump could be used as a factor in the measurement of the objective 'lower maintenance costs'. The cost however would have to be normalised on a scale 0-1 (in this case probably determined by £0 to the £theoretical maximum cost of spare parts). Proxy attributes - attributes that do not directly measure the objective - may be used where measurable attributes do not exist. For example, it would be difficult to measure the quality of an ambulance service. However, a measure of this quality is the time it takes for the ambulance to get to the patient. Obviously this does not measure the quality of care the patient receives when the ambulance arrives, but the time it takes to get to the scene can be measured and probably provides some indication of quality in respect that the patient is likely to be better off the sooner medical attention arrives. Keeney and Raiffa have written an excellent text which covers this topic in depth [Keeney, Raiffa 76].

Figure 9 depicts an example objective hierarchy that could be applied in pump selection. In this example the pump efficiency can be viewed as a general objective - to achieve as high efficiency as possible. Objectives are not usually achieved, they indicate general directions in which one should strive (in this case you can't get more than 100% efficiency therefore is the exception). Objectives that can be achieved are called goals.

Structuring the objective hierarchy

The initial step is to specify the broad objective. An objective generally indicates the "direction" in which one should strive to do better. For example, "to improve pump selection in heat transfer systems". The overall objective is too broad to provide insight into the different alternatives available to meet the objective. However. it provides a useful starting point to enable more detailed objectives to be specified in operational terms. For example, more detailed sub-objectives from the one identified above may be to "ensure consideration of pumping issues with regard to heat transfer system design". "ensure pump is cost effective". In turn, "ensure pump is cost effective" can also be divided into sub-objectives "reduce pump capital cost", "reduce cost of spare parts", and "reduce transport costs".

For the lowest level objectives one wants to associate an attribute that will indicate the degree to which attribute meets the objective. For example, "reduce pump capital cost" can be measured by the scalar attribute capital cost. Scalar attributes can be combined and represented as a scalar vector to represent the degree to which the higher level objective is met. The composite of a scalar vector is known as a vector attribute. Sub-objectives may conflict with one another, where the improvement of one will require some sacrifice on the part of the other. It may be possible that better solutions exist that improve both objectives simultaneously. If an objective cannot be directly measured, proxy attributes (see section 4.5.4) may be measured. It is important that each objective can be measured.

A formal method for structuring objectives is to build the model of the system under consideration, identify the relevant input and output variables, and the suitable objectives may become obvious. When dividing an objective into sub-objectives, care must be taken to ensure that all the facets of the higher level objective are accounted for in one of the sub-objectives. To ensure the hierarchy does not proliferate, the 'test of importance' can be applied. This ensures that a question is asked, 'could the best course of action be altered if that objective were excluded. The more an objective hierarchy is sub-divided, the easier it usually is to identify attribute scales that can be objectively assessed. When the hierarchy is limited, subjective measures of effectiveness must be resorted to.

Example pump system hierarchy

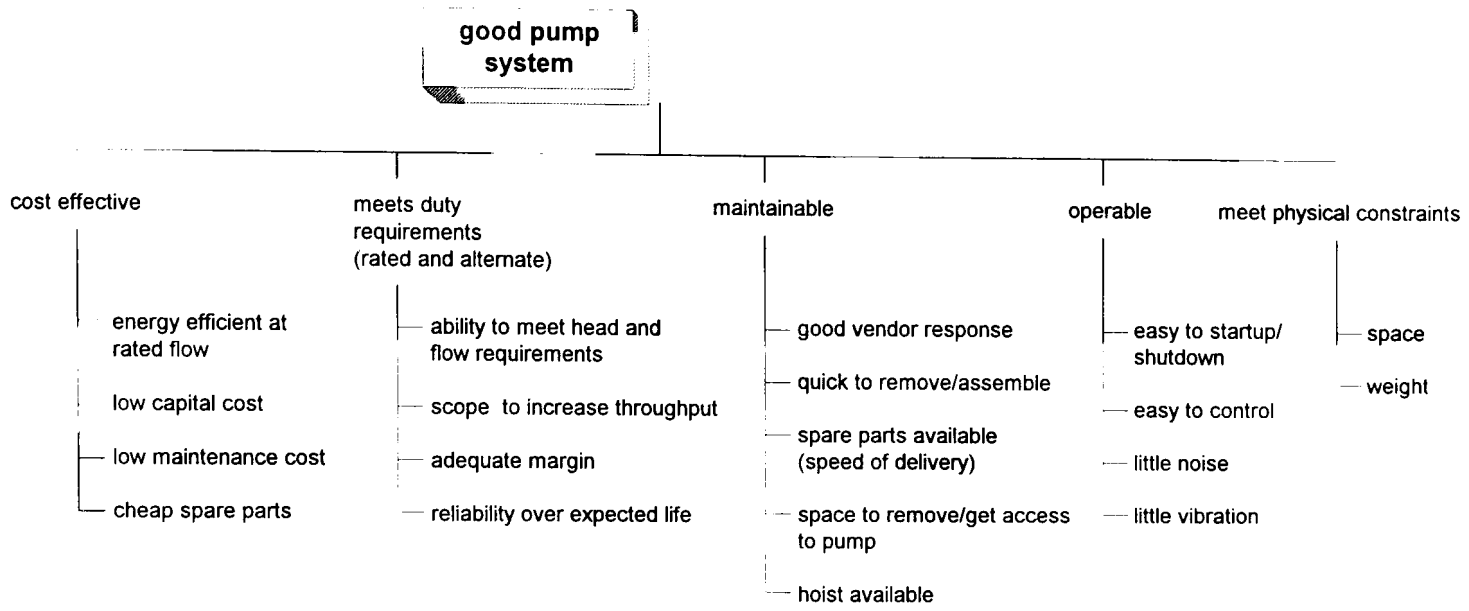


Figure 9 Example objective hierachy for evaluating a pump system

The 'good pump system' is a very general objective to satisfy. If the question is asked "what is meant by a 'good pump system'?" the reply may be that it should be cost effective, that it meets its duty requirements, it is maintainable, operable and meets the physical constraints on site (weight, dimensions etc). A formal understanding of an objective hierarchy will enable a formal decision analyses to be made concerning the selection of a pump for a particular application. This formal analysis is becoming more important given the increasing complexity in the number of considerations in the design process.

4.6. Framework topology

4.6.I Overview of the framework

The framework provides the network of communication channels required in a distributed multi-agent decision environment [Sage 90]. The framework to support the functionality required in the task model is depicted by an onion style model in Figure 10. The framework depicts the engineer on the boundary of the framework, supported in his task by an agent who has the capabilities to cooperate with other agents with different expertise in the environment.

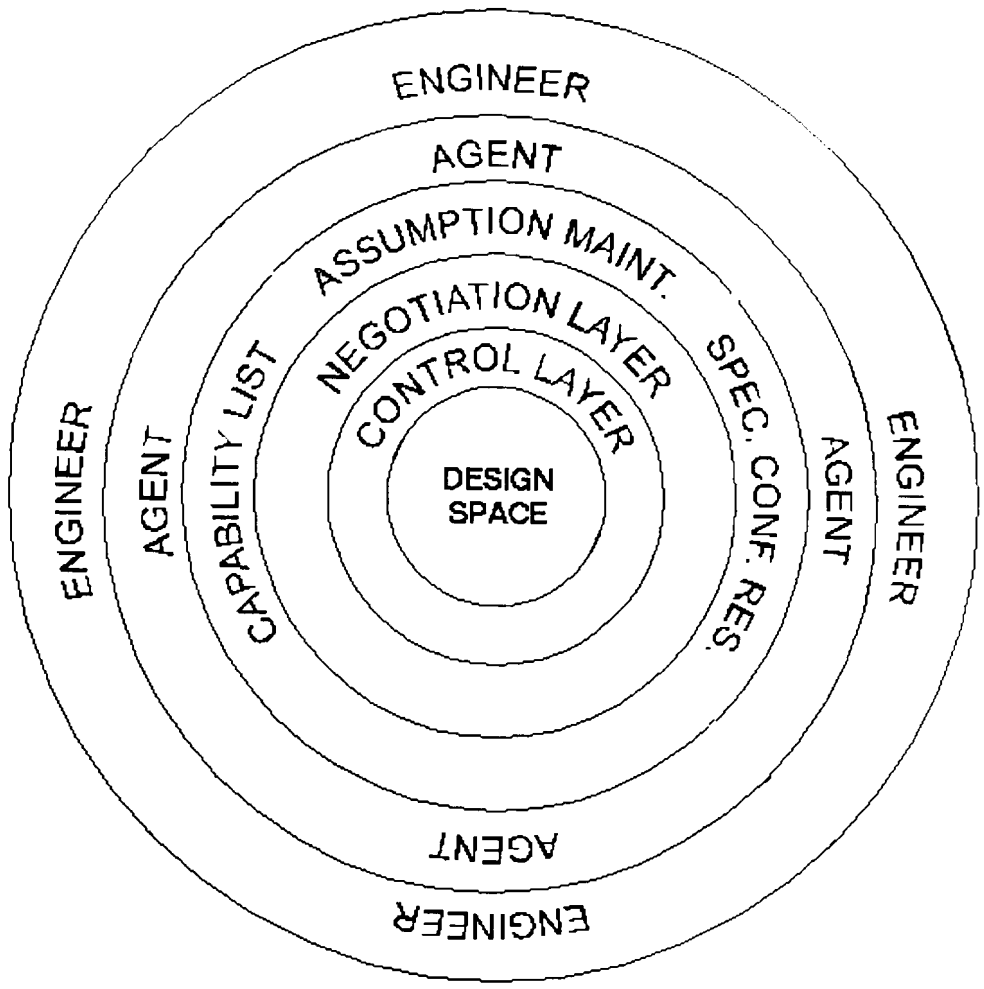


Figure 10 Concurrent Engineering Framework

The agent is a design program that has the capabilities to function independently from other design programs. The algorithms utilised by the agents may be in the form of simple design calculations, to some more complex artificial intelligent techniques and simulation programs. In order for the individual agents to collaborate with each other, they need to represent their capabilities and make assumptions. These functions are part of the 'enabling technology' layer. An additional function of this layer is to enable specific conflict resolution strategies to be specified that can be identified by the appropriate expert engineer. Adopting these strategies in the process of conflict resolution is likely to lead to acceptable solutions in a shorter time frame than any of the more generic un-informed strategies (e.g. search, compromise).

The function of the negotiation layer is to resolve conflict between the disparate agents. The negotiation layer takes control of the design process from the control layer when a conflict has been identified. When conflict has not occurred, or a specific conflict has been resolved, the control layer will be responsible for planning agent execution.

The design space is the central hub of the framework and holds all the shared design information. These layers are described in more detail in the following section.

4.6.2 CDEX Framework strategy

CDEX is based on a corrective strategy, although it has a preventive element. As mentioned previously (see section 4.3.2 - Framework Strategies), a corrective strategy will attempt to resolve conflict when it occurs. When conflict is identified in CDEX, the negotiation mechanism will manage the conflict resolution process. The preventive mechanism attempts to prevent conflicts before they occur. The approach to prevent conflicts in CDEX is modelled through the confidence assessments. If the confidence in a particular part of design is low, but the solution is considered good, then there is more likely to be conflict. Conversely, if the solution is good and the confidence is high, the agent believes that a conflict is less likely. CDEX has a conflict resolution strategy known as 'smoothing' (see section 5.4.1) which is a strategy that brings more information into the decision process in order to resolve conflict. This strategy is more appropriate where the agents are unsure as to whether the solution will turn out as good as expected (low confidence). This strategy is essentially a preventive strategy, in other words, a design cannot be accepted and proceed down a design route where it is unsure as to whether conflict exists.

The CDEX environment will enable design to progress without user involvement when provided with a design requirement. This design requirement can be to any level of detail the user can make available. In concurrent engineering engineers propagate the rational approach to design given a decision, and identify what the downstream issues are likely to be as a result of the decision. For example: an engineer determines that by reducing the total capacity requirement of a heat transfer system that is cooling a stream, he can save X pounds. The receiving tank is checked to ensure the temperature is ok and it can withstand the temperature. However, operations may determine that special insulation is required on the tank so that the operators are not scalded if they accidentally touch the tank. In this case, the insulation would cost more than the saving made by reducing the heat transfer requirement of the heat exchanger. This problem may not be recognised until late in the design stage when the down rated heat exchanger has been assumed to be ok in the design, and many other decisions have been made based on the assumption that the heat exchanger existed.

In CDEX, the initial requirement would be designed in more detail, and the 'rational' design approach determined from the down rated exchanger. The design would be produced in more detail, following a rational design approach, and the insulation would be a part of the result. This insulation may in the first stance be a warning to the engineer that there is now an additional design requirement to the initial design as a result of the decision to down rate the heat exchanger. If enough knowledge was populated in the system, a cost assessment could be made of both solutions and the engineer would find that the design with a reduced heat exchanger capacity had a higher cost which was not expected.

The example shown above shows that by enabling design to progress automatically through a rational design approach, one can implement a benefit normally associated with concurrent engineering where disciplines work together to identify the potential problems of a design approach. If CDEX did not make rational decisions in the design, the engineer may be working from incorrect advice and not make an accurate assessment of the design path, thereby missing an opportunity and selecting inappropriate solutions.

4.6.3 The engineer

There is a strong pull to enable the engineer to utilise tools that already exist and provide 'wrappers' to translate the software input and results into a form that can be shared by other agents (or software packages). The software packages currently available to the design engineers consist of packages such as CAD, process flowsheeting packages, sizing programs, and selection programmes. These packages can be neatly grouped into the types of engineering design classifications identified previously. For example, CAD performs a synthesis function, albeit by a human engineer. The engineer will be given a requirement to fulfill, and he will construct the required plant topology that is necessary to fulfill that function. The plant topology in this case will be classified as a synthesis process -the identification of components that can be combined to fulfill an overall objective. A process flowsheeting package can be considered as performing a sizing and synthesis function, as the major components of the process are specified (synthesis), and the requirements of each of the process units is automatically determined through simulation (sizing). The engineer can therefore utilise existing tools while being supported with advice from other systems available that account for issues covering the complete design process.

4.6.4 The agent

The ability to develop agents utilising different algorithms has remained a key priority. Engineering design utilises a great number of design techniques, and as yet there is no algorithm that encompasses the complete requirements of engineering design. The framework has therefore been developed to enable the different design algorithms and therefore currently available software packages to be utilised, while providing the facility to enable the different packages to cooperate in a generic design framework.

The agents are designed to act as single units in order to aid maintainability, as a single source can be utilised to maintain a single agent. Behind each agent may sit a traditional engineering package that provides a certain design function (e.g. CAD, Process Flowsheeting package). For the development of knowledge bases, the separation of concerns eases the knowledge elicitation task. If the framework is designed as a single large unit -rather than a distribution of units -then the design inconsistencies and possible conflicts have to be resolved during development. This not only means that the development will be an enormous task (as you will have to get group consensus on every

possible design issue), but that the engineering users will have to accept the output of the system as the ultimate arbitrator. In the cases where the engineer does not believe the proposal generated by the system, one is taking away the human responsibility (which has large implications), while additionally not enabling the engineer to learn from his understandings by promoting discussion with other engineers.

A distribution of agents will enable more computational power to be applied in resolving the design problem, as solutions can be developed across multiple processors and the results integrated. The distribution will also promote a more incremental development approach to the system which is more appropriate given the enormity of the task to develop a system that encompasses the complete life-cycle perspective.

4.6.5 The negotiation layer

Distributed environments fundamentally require the adoption of a standard protocol and standard data representation. These are fundamental requirements for engineering support applications to share information. Expert systems that support the concurrent engineering design philosophy however have an additional requirement. The knowledge based systems are expected to cooperate and present a 'shared response' to a particular design problem. Neither party in engineering design can achieve their goals unilaterally, and therefore cooperation is required in order to reach a solution [Anson, Jelassi 89]. It is important to ensure that the different design perspectives are reviewed and appropriate design trade-offs carefully balanced in order to develop a solution acceptable from the global perspective.

The ability to trade off the different engineering concerns to arrive at a shared solution is the function of the negotiation layer. The negotiation layer will maintain an objective hierarchy, detailing the objectives and trade-offs from the global perspective. This objective hierarchy will be built up from a collective understanding of the engineering disciplines. This is in line with the concurrent engineering principle of sharing life-cycle goals. Reality however suggests that engineers are unlikely to agree fully with the solutions proposed by such a framework. This suggests that either the engineer does not fully appreciate the trade-offs made in the objective hierarchy, or that the design knowledge base is not complete and some important factors have not been considered. From analysis of the requirements, the engineer must be allowed to maintain control of the design. The aim is to provide design support, and not to impose a computationally derived solution onto the engineer.

The objective of the negotiation layer is to resolve conflict. Initially, a review will be performed of the specific conflict resolution strategies defined with the agents. If one is appropriate it will be applied to resolve the conflict situation. It has been identified that specific conflict resolution strategies can be determined at the knowledge elicitation stage and are more likely to lead directly to a solution and be more widely acceptable [Klein, Lu 89].

If there is no specific conflict resolution strategy, a more generic approach will have to be applied. Initially the conflict must be analysed to determine what type of conflict resolution strategy is appropriate. This analysis will determine:

i. how *close* the parties are to resolving conflict,
ii. the *number of issues* involved,
iii. *flexibility* on issues,
iv. agent *capabilities*,
v. *actual* or *anticipated* conflict,
vi. if conflict is over the *effect of an alternative* or the *goals*,
vii. the *importance* of reaching an agreement, and
viii. the *risk* of the alternative strategies in computational terms.

From this analysis a conflict resolution strategy is selected that is most appropriate for resolving the conflict in reasonable time and with minimum interaction with the engineering user. For example, a compromise resolution strategy is appropriate where the difference between the conflicting parties is small (*close*) and the agents are flexible over the issues involved (*flexibility*). However, a compromise solution is a solution where both parties have had to make a sacrifice, and they both lose. An alternative that is not acceptable to the conflicting parties does not discount the fact that a better alternative is available that may be acceptable for all conflicting parties. Obviously you will have to account for the computational effort required in searching for another solution (*risk*), but if the matter is critical towards meeting the required objectives (*importance*) -may be the cost of search is small. As you can imagine, the selection of an appropriate strategy is not simple and can be viewed itself as a knowledge based problem [Lander, Lesser, Connell 90].

4.6.6 The design space

The communication method adopted in a distributed environment will fall into one of two categories, message passing, and through the use of a design space with characteristics of a blackboard [Rich, Knight 91]. Message passing systems enable disparate software systems (or *agents*) to directly request and post information to each other. Obviously in this type of framework, agents are expected to know the explicit names of the other agents and their capabilities.

A **design space** mechanism is more appropriate in a concurrent engineering framework. In large complex engineering environments there are likely to be many different agents, and maintaining knowledge of the links to other agents will be prohibitive. The design space provides the means through which all agents can communicate with each other through a common knowledge representation format. The common data representation format in an engineering environment should follow the standard defined by STEP (Standard for the Exchange of Product Model Data [King, Norman 92]).

The design space provides a more comprehensive function than that provided by an engineering database. The design space not only maintains a list of agents interested in the evaluation or design of particular objects, but also maintains a list of dependencies between the design objects and the requirements on which they are based. For example, if a pump design is based on design detail pertinent to an inlet vessel (which it obviously would be), any information modified regarding the inlet vessel will require a re-assessment of the pump design. The design space therefore keeps track of which proposals, conflicts and evaluations are based on which particular design object. When the detail of a particular design object is changed, any design, evaluation, or conflict, based on that information must be re-accessed.

There are however problems associated with a central data model. In essence it is similar to the development of a standard database model - or 'entity relationship' model - that can underpin many different systems in an organisation. An advantage with these types of 'shared' databases is the maintenance of consistency and the avoidance of re-keying information into different corporate systems. The problems is that there is no single way to model the problem, and getting the right representation to suit the many different views and tasks is a difficult problem. Changes to a data model late in the design stage can cause considerable amounts of knock on work which requires changes to potentially many of the systems components. Once the model is working however and appears effective, models are not difficult to extend.

The data model chosen by CDEX is based on an object oriented style for reasons outlined in the next chapter. Issues concerning data collection and classification of data has been studied by Sage [Sage 90]. Sage has identified many ways in which information can be categorised: accuracy, precision, completeness, sufficiency, understandability, relevancy, reliability, redundancy, verifiability, consistency, freedom from bias, frequency of use, age, timeliness, and uncertainty. The main concerns highlighted by Sage's research are that,

i. information should be presented in very clear and familiar ways, such as to enable rapid comprehension

ii. information should be such as to improve the precision of understanding of the task situation

iii. information that contains an advice or decision recommendation component should contain an explanation facility that enables the user to determine how and why results and advice are obtained

iv. information needs should be based upon identification of the information requirements for the particular situation

v. information presentations and all other associated management control aspects of the support process should be such that the decision maker, rather than a

computerised support system, guides the process of judgement and choice.

These requirements are sometimes difficult to meet. Explanation facilities (iii) are difficult to build as computers often reason in ways different from a human and therefore mapping a computational result onto a reasoning path that a human will easily follow is a complex problem. Point iv, where 'information needs to be based upon identification of the information requirements for a particular situation', does not help too much in the case where a generic framework is being built and where a complete understanding of all future requirements in not known. This last point has a greater impact on the design of the framework as opposed to how the data is represented. The point stresses one of our requirements of the framework highlighted earlier, and is therefore one that has already been considered.

4.6.7 The control layer

The function of the control layer (CL) is essentially to plan and direct the execution of the agents. It requests designs and design review capability from each agent that has highlighted its interest in particular aspects of the design problem. This ensures that the results and advice produced by the system account for all the knowledge maintained. Secondary to this, the efficiency of the CL is measured by the efficiency at which it arrives at a solution. Efficiency is determined from a number of factors including time, requests from an engineering user, and processing effort. From the concurrent engineering viewpoint, one wants to avoid going down a path that is likely to be undesirable, and identify this at an early stage without too much computational expense.

There are essentially three methods for controlling the execution of agents. Firstly a plan or event sequence that identifies the ordering in which the agents are requested to perform the design tasks is required. This plan can be followed until it fails, for example, an agent cannot continue because the requirements cannot be met. This 'planning' strategy has little computational effort unless the plan has to be re-designed in order to cope with any problems found.

A second method for controlling the execution of agents is where the controlling mechanism determines which agent can have an input dependent upon the state of design and the capabilities of the agents available. For example, if an agent produces a requirement for a pump, then an agent that is capable of selecting a specific type of pump is invoked. This mechanism is more flexible and is not affected by plan failure. The method also has the advantage of being flexible in the introduction of 'new' agents. Once the new agent has represented his interests, he is requested to perform work when the need arises. A new plan does not have to be devised.

The third approach to agent execution is to enable the agent to run at every point in the design process. This is a computationally expensive option as an agent will be requested

to review or put forward an extension to the design in many cases when his input is not necessary. The approach is analogous to a human workgroup where a person can have an input to any stage of the design process where he sees fit.

From analysis of our requirements, a combination of both the second (registering of interests) and third (continuous review) is required. The registering of interests is possible as an agent can only perform a certain set of tasks and these can be written down explicitly. The continuous review would be expensive in resources. The planning approach is susceptible to conflicts and 're-planning' and therefore would also present an expensive option. There is a requirement however to include the views and interests of potential human agents, and therefore an ad-hoc presentation of ideas and conflicts at various stages of the design process is possible. This requires some aspects of the continual review process where issues can be raised at any level of the design process and at any time.

The control layer looks after the 'day to day' operation of the design process - when design is progressing without problems. When conflict is identified however, control is passed over to the negotiation layer which is responsible for directing effort where it considers is best for resolving the conflict, accounting for both processing effort and solution quality. The control layer therefore relinquishes control to the negotiation layer when conflict is identified

4.6.8 The enabling technology layer

For an agent to participate in an environment where the response to design requirements is collective, mechanisms are required to :-

i. ensure that the agent capabilities are represented,

ii. any specific conflict resolution strategies are in a format appropriate for application,

and

iii. that the agent is able to participate in the design with little design information.

Capabilities

In order for agents with different capabilities to cooperate in a design environment there is a need to have some way of representing their capabilities. In the CDEX (Concurrent Design Expert) environment the agents initially present their interests in the design space. These interests are recorded and the agent is requested to perform certain functions as and when required by the central control layer or the negotiation layer. Interests are recorded by a *design function* and *evaluation function* together with the corresponding design object. The design/evaluation functions are those identified previously (synthesise, select, and

parametise) and the design objects (e.g. pump, vessel, pipe, valve) are dependant upon the objects specified in the object hierarchy.

Engineering assumptions

In the early design phase very little information is available. This presents the problem of how to ensure a life-cycle perspective is achieved when most of the design agents can not participate due to a lack of information. This problem can be alleviated through the use of assumptions and constraints. Without information, the agent can assume a value of a design attribute that may be the most credible. Assumptions, due to their nature, are obviously the first area for analysis when they lie on a conflict path.

Specific conflict resolution strategies

It has been identified that conflict resolution strategies can be determined at the knowledge elicitation stage [Klein, Lu 89]. Specific conflict resolution strategies are appropriate for two reasons. Firstly they are more likely to lead directly to a solution rather than applying some iterative technique to determine the problem and resolve conflict. There is therefore a computational saving with this approach. Secondly, the solution is more likely to be immediately acceptable, as specific resolution strategies identified at a knowledge elicitation stage are more likely to be the norm, and a more acceptable way of resolving conflict with respect to other design considerations.

The framework therefore requires a mechanism for representing a conflict resolution strategy.

4.7. Summary

This chapter has addressed the requirements and design of a framework that can provide the basis support to enable engineers to review design decisions in the light of the life-cycle requirements. An example 'model of use' was developed detailing how an engineer would use the CDEX system to support the design task. An important concept in the framework is the ability for integrating the expertise and enabling agents to negotiate and collectively present advice to the engineering disciplines. The objective is to provide an environment where the engineer considers the global perspective in the design decisions made.

Utility weighting and the 'objective hierarchy' are the formal techniques applied that enable the users needs and values to be represented in the framework. Utility theory may appear a rather simplistic method of assessing designs for appropriateness. It is however a very powerful one and has been used in many design systems. Different methods of

ranking exist but they all work on the basic premise that a particular design item can be identified by a certain set of numeric attributes, and that these attributes can be used in one form or another to measure the effectiveness of a solution.

In order for a single design to be developed, there is a requirement for a standard data representation so that disparate software systems can communicate design intent and understand the designs developed by other external agents. CDEX is a data driven approach to design, for example, when a particular design object is presented for say a pump, those agents interested in pump design are requested to review the pump and provide additional design detail. In order to enable data driven design, an understanding of the basic design processes was required. The CDEX mechanism relies on three different types of engineering design classification: 'selection', 'synthesis', and 'parametric' design [Kannapan, Marshek 92]. This classification appears to encapsulate the basis of the design requirements in the engineering design of process plant and therefore was the approach applied in CDEX. Object oriented principles were used in the development of the framework due to the effective analogies between the technique and the basic design principles: the 'is-a' relationship and design selection, and the 'part-of' relationship and design synthesis.

This chapter has represented CDEX through a number of layers. The negotiation layer is an important area of development where there are disparate viewpoints, as in the case of engineering design teams. Considerable efforts have been made to improve other areas of knowledge representation and communication. In contrast, efforts spent in the development of negotiation mechanisms has been limited. The next chapter identifies the issues in negotiation and discusses the potential strategies to resolve conflict.

Chapter 5. Negotiation and Conflict resolution

5.1. Overview

Parallel working by all life cycle perspectives early in the design process is an important facet of concurrent engineering. Engineers must collaborate effectively in order to understand each others' requirements and ensure the optimum design with respect to all design viewpoints. Parallel working in the early stages of a project is more important due to the larger grained decisions that have to be made. Early decisions set the lower bounds on cost, time complexity and risk, as well as establishing the upper bounds on achievable product reliability and customer satisfaction [Cleetus 92].

The partitioning and overlapping of design tasks however has the inevitable penalty of inconsistency [Cleetus 92]. The interdependencies, problem dimensions, issues, and different criteria of the individuals are the inevitable cause of conflict which has to be resolved.

In order to avoid the problems with traditional knowledge based systems, an approach to partitioning expert systems into loosely coupled systems - termed 'Agents' - has been proposed [Uma, et.al. 93]. Having single autonomous agents however poses the need for coordinating them towards solving complex design problems. Reasonable models of distributed expertise will dictate that agents hold incomplete local views of the problem [Davis, Smith 88]. Agents may possess knowledge unknown to other agents, maintain different beliefs and evaluation criteria, have incompatible knowledge representations, or may just be logically inconsistent with each other [Bond, Gasser 88]. Conflict is therefore inherent where expertise is distributed.

There is a variety of conflict resolution strategies that can be applied to resolve a conflict situation, both in the human workgroup and between disparate knowledge based agents. The strategies all have their strengths and are appropriate in different situations. Negotiation has been proposed as a conflict resolution strategy in the sense that the roots of conflict are examined and rectified during negotiation [Pruitt 81]. In artificial intelligence terms, negotiation has been viewed as a conflict resolution and information exchange scheme [Bond, Gasser 88]. The computational form of negotiation has been considered as a high level conflict resolution protocol, that identifies the key problem, maps the problem domain, and iteratively applies appropriate conflict resolution strategies until the conflict is resolved. In this sense the negotiation mechanism plays a coordinating role, rather than sitting alongside techniques such as compromise, constraint resolution, assumption surfacing and other conflict resolution techniques.

In this chapter the requirements for a computational negotiation mechanism are explored. Additionally a review of the nature of conflict is made, and how conflict is recognised, and

the various mechanisms that can be applied to resolve conflict. Conflict resolution techniques applied in the human workgroup are documented in order to develop an analogy and promote the development of computational techniques.

5.2. Conflict

The Nature of Conflicts

A conflict is a disagreement between two or more viewpoints on some decision or value proposed in a design. These disagreements are a result of the differing *needs* that lead to incompatible preferences among the alternatives under consideration [Pruitt 81]. Needs comprise both *goals* and *values*. 'Goals' are the end states to which a viewpoint is moving and reflect some underlying values. 'Values' are the events or states of nature that a party finds particularly appealing or distasteful [Pruitt 81] and cannot always be translated into goals at any one time. An example of a difficult decision based on the different needs involved is highlighted in Figure 11. These differing needs however are not the inherent cause of conflict. It may be possible to find a solution to a conflict where all parties are satisfied with the outcome.

GEC gains height for £2bn target by Roland Gribben

GEC and its American partner yesterday increased the stakes in the three-way battle for a £2 billion helicopter order by offering an improved version of their entry and holding out on the prospect of a £500m export spin-off.

Lord weinstock, managing director, also disclosed that the consortium had opened discussions with Westland, the GKN helicopter subsidiary, about assembling its entry, the Cobra Venom.

Westland is in partnership with McDonnell douglas, the American aircraft manufacturer, for the order with the Apache helicopter, and is seen by some as the front runner in a battle now moving to a climax.

British aerospace is the third entry through a tie-up with French and German companies with the Tiger.

GEC and its partner Bell Helicopter Textron had been regarded as back benchers in the three-way fight to provide the army with 91 attack helicopters. But GEC believes its prospects have been boosted by the offer of a four-blade rotor version of the Cobra Venom and the chance of American business.

Under the existing plan GEC will provide the electronics for the army order but if the defence ministry plumps for the Cobra Venom, the American defence department is hinting it will be prepared to offer the British equipment for its existing fleet of 220 Cobras.

Lord Weinstock said discussion between the British defence ministry and the pentagon in America could result in the signing of a memorandum of understanding similar to that for the Bae Harrier jump jet programme and help underpin the long-term future of the UK avionics industry. GEC estimates that production of the Cobra Venom could offer 14,000 jobs in Britain.

The defence ministry is said to be supporting GKN and the Apache, Michael Heseltine, trade and industry secretary, the Tiger because of its European credentials, while the treasury is believed to prefer the Cobra Venom because it could be £5m cheaper.

Figure 11 Extract from the Daily Telegraph, June 7th, 1995, highlighting the potential difficulties in making a decision where multiple concerns are involved.

Conflicts over the alternatives

For example, a cost engineer may want the cheapest pump possible for a particular job. On the other hand, the process engineer wants a pump that is most technically capable of performing the job. The two engineers are not in direct conflict over the issues, the cost engineer may not care about technical elegance, and the process engineer is not too concerned about cost[2]. However, if only a few solutions have been explored, it may be the case that the set of pumps preferred by the cost engineer is mutually exclusive to the set of pumps preferred by the process engineer. In this case they can either select a pump

[2] Please note that this is a hypothetical example to highlight an important point.

from either set through making some sacrifices on their ideals, or they can look for a pump that is both technically elegant and cheap. Obviously looking for another pump that is both cheap and technically elegant can lead to a better solution, as engineers making sacrifices will reach a solution that is not optimal to either of them. However, the cost of searching for other pumps that are both cheap and technically elegant may take a considerable amount of effort, and ultimately may not exist anyway.

The concept that conflict is based on incompatible alternatives is important and promotes emphasis on the search for alternative solutions that can satisfy all parties. The tradeoff between the cost of search and changing the individuals ideals is a problem that must be addressed in negotiation.

Anticipated conflict

Conflicts do not just occur on disagreement on some decision or value. Conflicts can also occur when a party identifies that due to some decision or constraint a possible conflicting situation can be foreseen sometime in the future. These particular conflicts have been defined as 'anticipated conflicts' [Klein, Lu 89] and may be handled differently from actual conflict. Actual conflict is where a viewpoint has proposed a value or solution that is in direct violation of a constraint or opinion of another viewpoint. Anticipated conflict is where a viewpoint has proposed a value that restricts the other viewpoint from either making a feasible consistent design, or restricts the decision optimality of the other viewpoint. It is important to differentiate between these two types. Concurrent engineering promotes the sharing of life cycle perspectives and aims to avoid conflict in later design stages through early consideration of the issues. There is therefore a higher degree of anticipated conflict than that in the traditional sequential design process.

Run-time vs. development time conflicts

When developing knowledge based systems there is the option of integrating or defining knowledge so that in normal use conflict does not exist. In other words, conflict resolution is performed at development time. However, in a distributed concurrent engineering environment with many complex and interacting agents, ensuring consistency between knowledge bases before use would be prohibitively expensive. The alternative is to resolve conflict between agents in operation, this is termed 'run-time' conflict resolution [Polat, et.al. 93] and has many benefits:

i. the development and maintenance of different bodies of expertise in isolation
ii. reduction in development time due to not having to perform consistency checking between many different bodies of expertise
iii. enable new bodies of expertise to be added and to maintain their independence
iv. maintain flexibility in the type of conflict resolution strategy used
v. involve human problem solvers in the process of conflict resolution

Maintaining these benefits, especially the natural interaction of human problem solvers, is crucial in a concurrent engineering environment.

5.3. Stages in the resolution of conflict

5.3.1 Stages in the human workgroup

There are identifiable stages in the human workgroup to enable the resolution of conflict. A concurrent engineering system is expected to explain and justify its results to a design engineer and therefore knowledge of these stages is valuable. The problems identified in the human workgroup can also aid in the development of knowledge based practices to reduce these problems.

Agree a problem exists

Obviously before a conflict can be resolved, conflict has to be recognised. This is not a simple problem. Direct comparison of alternatives or disagreement with an alternative are the obvious direct form of conflict. Conflicts that are due to some deviation in standards means that some standards need to be in place. The development of these standards may itself be a highly controversial activity and additionally may be a cause of conflict [Gray, Starke 84].

When a party has identified a conflict situation, other parties with an interest over the values in conflict will have to agree that a problem exists. Since different parties have different perceptions of how significant a problem is, the conflict resolution issue is again complicated.

Understand the problem

When there is agreement that conflict exists, the parties should then come together to define the problem exactly. Many parties attempt to resolve conflict without adequately understanding the actual decision or value over which there is conflict. Without understanding the problem both parties will become more frustrated and resolution of the conflict will take much longer. De Bono [De Bono 89] provides techniques for helping people to obtain an adequate map of the problem area. These techniques cover an examination of both sides, analysing the other person's views, and systematically listing the points of agreement, disagreement and irrelevancies. From this map, one has to pinpoint the variables that are in conflict.

The root of conflict

Analysis of the values in direct conflict may not be where the basis of the conflict rests or where a solution is to be found. Values may have been inferred from previous

assumptions, that may be incorrect, or from previous design decisions that did not account for all design viewpoints. There may be a long string of possible causes for the conflict identified. A common problem is to already have an idea of the cause of the problem and look for the evidence to support it. If the root of the problem lies along another path, the person is unlikely to find it [Gray, Starke 84]. Apportioning blame to a primary cause of conflict, where a number of possible causes exist, is itself a credible problem.

Generate Alternatives

After the problem has been clearly specified by the conflicting parties, an attempt should be made to generate some possible alternatives. People tend to have difficulties performing this task. People tend to generate a couple of alternatives and then start to identify the positive and negative aspects of each without sufficient thought [Gray, Starke 84]. People also tend to suggest things that have worked in the past, and the 'we haven't done that before' attitude is prevalent. There is therefore little scope for innovation. In these cases it is unlikely that the parties will develop a solution that is optimal with respect to both of their ideals.

Evaluate and Select Alternative

After the alternatives have been generated they have to be evaluated. In this case, a criteria for evaluating alternatives is required in order to determine which alternative is best. Individuals usually avoid this step, and if they do make an attempt, it is usually a shallow analysis because they are not motivated to spend the time or the effort that is required to make the decision properly [Gray, Starke 84]. The best alternative must then be chosen from the alternatives generated and the selected criteria of evaluation.

5.3.2 Stages in the computational approach

Procedures to manage, scope and select the conflict resolution process vary.

Initially, one or more agents must present a proposal. There may be confidence limits attached to the proposal if the solution was derived from some assumed, or non-exacting evidence [Polat, et.al. 93].

Methods differ in how the proposal is reviewed by other design agents. In Sycara's model of negotiation [Sycara 89], if a proposal is rejected, an attempt is made to repair the proposal using multi-attribute utilities and constraint satisfaction techniques. A counter proposal is then produced, together with the justifications for the proposal. Another approach [Lander, Lesser, Connell 89] was to integrate any conflicting proposals using conflict resolution strategies. The Cooperating Experts Framework (CEF) [Lander, Lesser, Connell 89] brought together all conflicting parties into a 'conflict set', and the result was to either revise or abandon the proposal.

Computational methods differ in how conflict is resolved. There are two different dimensions to this problem, one dimension is the concept of globally shared goals versus the idea that agents can maintain their own beliefs and goals. The other dimension encompasses the use of generic conflict resolution strategies versus domain specific.

The concurrent engineering ideology is towards shared goals and team ownership of products. In complex domains however, individuals hold different beliefs and viewpoints and there is no concept of globally shared goals. The development of global evaluation criteria is elusive in terms of the difficulties presented in knowledge elicitation, development, and maintenance. Individuals may hold different interpretations of any goals specified which pervades a global interpretation. An approach being considered is to enable the specification of global goals, and to provide the facility for engineers to compromise these goals if it is considered necessary. In this event it can be assured that the life cycle perspective has been provided while enabling the engineer to utilise his own experience and maintain responsibility for the design.

Considering general conflict resolution strategies, whatever the size or importance of the parties, much the same causal principles seem to apply. This suggests that it is possible to talk about a general theory of negotiation [Pruitt 81]. General conflict resolution strategies exist [Polat, et.al. 93][Klein, Lu 89] and can be used in the resolution of conflict. The advantage of general conflict resolution strategies is that agents, and the addition of new knowledge can be placed in a negotiated environment without having to maintain specific conflict resolution strategies. Specific conflict resolution strategies are domain specific and can be derived from the knowledge elicitation exercise [Klein, Lu 89][Lander, Lesser, Connell 89]. These strategies have the advantage that they are more likely to lead to an acceptable solution to the conflict in a shorter time, primarily by reducing the search. Polat et al [Polat, et.al. 93] has developed a hierarchy of conflict types, the general types at the top and domain specific strategies at the base. If conflict occurs, search for a strategy proceeds from the base of the hierarchy, and if no domain specific conflict resolution strategies exist, the more general ones are utilised.

There is no one single strategy that is best for resolving conflict situations, and the selection of an appropriate strategy can itself be viewed as a knowledge based problem.

5.4. Conflict resolution mechanisms

5.4.1 The basic strategies

There are three basic conflict resolution strategies identified by Pruitt [Pruitt 81] from his research into human workgroups, cooperative behaviour, competitive behaviour and unilateral concession.

Engineering design is a complex activity requiring input from many design disciplines.

All viewpoints cannot achieve their goals unilaterally, and therefore cooperation is required. Cooperative situations typically involve conflict resolution techniques such as compromise, the abandonment of less important goals and consensus, where the aim is to find as mutually beneficial a solution as possible [Klein, Lu 89]. In order to support collaborative behaviour, the bargainer must have the goal of achieving coordination and trusting the other viewpoint [Pruitt 81]. If the other viewpoint is not willing to coordinate then there is a risk of being exploited. Cooperation is a vital ingredient in concurrent engineering.

Competitive behaviour is common where there is mistrust of the others' intentions, or where one viewpoint has threat capacity over the other. It is where one viewpoint seeks to gain an advantage at the others' expense [Pruitt 81]. This type of behaviour is wasteful in the sense that parties spend considerable time attacking and defending their own viewpoint [De Bono 89]. Competitive behaviour cannot be seen as advantageous in a concurrent engineering environment where the attitude is towards shared goals and team ownership of a product. Individuals in this environment have nothing to gain from competitive behaviour, as the team is judged on the quality of the end results, rather than on the individual's performance.

Concession making is an alternative strategy to competitive behaviour in that whatever encourages one strategy, discourages the other. The nature of competitive tactics is to encourage the others' concession making [Pruitt 81]. A concession is defined as 'a change of offer in the supposed direction of the other parties interests that reduces the level of benefit sought' [Pruitt 81]. Concession making therefore reduces the level of conflict between the parties.

Cooperative, competitive and concession making can be seen as the basic strategies for moving towards agreement. Conflict resolution strategies can be classified into one of these groups.

5.4.2 Strategies applied in the human workgroup

There is a wide variety conflict resolution strategies in the human workgroup. There is no single strategy that is appropriate to resolve all conflicts and the ones covered below are appropriate in different situations.

Compromise

Compromise is a popular strategy where a solution is reached by both parties sacrificing some of their goals or values in order to reach agreement. This is a good solution to major conflicts where both parties want the other parties to concede something, or where the differences are not too severe and can be modified in order to suit both parties. The two major problems with compromise are that neither party gets what they want, and that people may exaggerate their demands in order to have something to sacrifice [Gray, Starke

84][De Bono 89].

Integrative

An integrative strategy is the search for the win-win situation, where both parties collaborate to develop a solution that matches both their needs [Gray, Starke 84]. It is appropriate where compromise solutions can not be identified or where the differences are so severe that a compromise solution will not be adequate to either party. An integrative strategy is only appropriate when the conflict has more than one dimension [Pruitt 81]. Single dimensional conflicts in which there is only one issue, are not suitable for integration as the sum of any parties benefits is the same for every alternative. This is commonly known as zero-sum, where one parties gain is another parties loss [Pruitt 81]. This situation is relatively uncommon, and most alternatives have 'variable value' [De Bono 89]. Value can differ according to the person and the situation [De Bono 89] and what may seem of little importance to one party, may be of considerable importance to another.

Consensus

Consensus is when both parties work together to find the best solution to their problem. It is appropriate where the issue under consideration is so important that the conflicting parties can easily see that the issue must be resolved [Gray, Starke 84]. Consensus is seen as a way of bringing harmony among conflicting requirements and ends in a sense of group achievement which overcomes the individual's loss from having to yield along the way [Cleetus 92]. Consensus however means staying with a part of the proposal on which everyone is agreed, it is a passive approach [De Bono 89] and does not offer the advantage of search for other solutions.

Confrontation and Smoothing

To aid in reaching agreement it is appropriate to have a good understanding of the domain. This includes understanding the problem and the viewpoints of the other parties in conflict. Conflicting parties may often fail to state their real problem [Gray, Starke 84] which obviously causes difficulties for other parties who are trying to resolve the issue. These difficulties can be overcome by adopting a 'confrontation' resolution strategy, whereby each party in conflict is initially required to state their problems and viewpoints. Where the conflict is due to a lack of information, a 'smoothing' strategy can be adopted. Smoothing is the addition of new information to a conflict in order to resolve it. Smoothing is not appropriate however where the conflict is due to a fundamental difference in values.

Exlectics

De Bono has proposed exlectics as a method to pull out the items of value from both sides

of the argument. Initially there are no opposing or varying ideas, just joint listening and joint exploration of the problem space [De Bono 89]. There are a number of tools to explore and map the problem space, therefore enabling more constructive consideration to the conflict. These tools cover the analysis of the parties agreements, disagreements and irrelevancies, and examining the problem from the other parties point of view.

Forcing and majority rule

Other popular resolution strategies are 'forcing' and 'majority rule'. Forcing is where the resolution of a conflict is 'forced' by another party, say a manager. This is unlikely to be the best solution as the manager is unlikely to have the thorough understanding of the problem domain as the parties in conflict. This may also lead to resentment from the party that was required to concede and may lead to the loss of his or her support. 'Majority Rule' is where a group votes on a solution. Although this approach is democratic, research has shown that a quick vote on issues usually suppresses thoughtful consideration [Gray, Starke 84].

5.4.3 Computational strategies for resolving conflict

From understanding human conflict resolution behaviour, an insight into computational conflict resolution methods can be formed. The methods highlighted here cover the area of conflict resolution at run-time, due to the inherent problems in development time consistency checking [Klein, Lu 89].

Compromise bargaining

Launder and Lesser [Lander, Lesser, Connell 89] have developed a mechanism for 'compromise' bargaining based on Pruitts work. They suggest that it is suitable where the level of disagreement between the conflicting parties is small. In order for compromise bargaining to proceed there has to be identifiable constraints that can be relaxed, and a mechanism available for relaxing the constraints. When a compromise solution is adopted however, both parties lose. A conflict where the level of disagreement is small, does not preclude the fact that a better alternative is available. However, when such small disagreements are apparent, the computational expense of searching for further solutions may not be justifiable.

Integrative strategy

Where there is a large difference of opinion between conflicting parties, an 'integrative strategy' is appropriate [Lander, Lesser, Connell 89]. Re-evaluation of the goals and assumptions of the design may develop a solution that is more appealing to both parties.

Case based reasoning

Case based reasoning can be seen as a method for improving the efficiency of a conflict resolution process. By remembering complex negotiation steps for previous successful resolution of conflict, the search for a solution can be avoided. Case based reasoning has been suggested in negotiation where there are multiple conflicting parameters, and the dependencies between them are not well understood [Sycara 89]. From reviewing case based reasoning applied by human workgroups in design [Gray, Starke 84], a couple of problems can be identified. Case based methods do not normally generate innovative solutions, and therefore improved negotiation strategies may be missed. Also past cases for solving conflict may be useful, but may not really confront the problem at hand. The biggest problem in computational case based reasoning strategies is mapping the problem to its existing cases.

Constraint resolution

Conflict can be identified by the violation of constraints. Conflict can therefore be resolved by the 'relaxation or dropping of constraints'. In order to relax constraints, you need to be able to identify whether constraints can be relaxed and have methods available for relaxing the constraints. The degree to which constraints can be relaxed, or dropped, is of course subject to the individual agents preferences and success criteria.

Numerically weighted constraint resolution has had some successes, although it suffers from the common problems faced with other numerical weighting techniques [Bond, Gasser 88]. These problems cover the acquisition of numerical weights from human experts, their use in explaining results, the difficulty in changing system behaviour (due to the large number of factors), and maintaining consistency and the context applied by different experts.

Super agent

Conflict between agents can be also be resolved by some 'higher level' agent or some 'agent in authority'. By higher level, it is meant that the agent has some general knowledge of the domain in which both agents are in conflict. The problems with this type of super agent are similar to the basic problems in developing single large expert systems which distributed environments are attempting to overcome [Lander, Lesser, Connell 89][Uma, et.al. 93].

Sharing knowledge

For conflicts that result from lack of knowledge, conflicts can be resolved through 'sharing knowledge' (similar to *smoothing*) that results in agreement. This form of sharing knowledge however imposes the constraint that the agents in conflict share a common knowledge representation, or that a suitable translation mechanism exists. Sharing

evidence that supports particular hypotheses (*evidential argumentation*) can also be used to reach agreement.

Conflicts between agents may be due to different assumptions, and not on issues and goal criteria. In this case some form of backtracking is required to locate and resolve these conflicting assumptions. 'Assumption surfacing' is the underlying technique found in many forms of negotiation [Bond, Gasser 88], and is the basis for truth maintenance systems.

5.5. Selection of a strategy to resolve conflict

The selection of an appropriate conflict resolution strategy in any given situation is dependent upon the characteristics of the problem domain, characteristics of the conflict resolution strategy, agent capabilities, and the status of negotiation.

Dimensions of the problem domain

The problem domain can be highly complex, especially in a concurrent engineering environment with many issues and many agents involved in design. There are numerous dimensions in the problem domain that can effect the selection of an appropriate conflict resolution strategy. These dimensions may individually or collectively determine an appropriate strategy or mechanism to choose:

i. How close the parties are to resolving the conflict
ii. the number of issues to be resolved
iii. Agent flexibility on issues
iv. Capabilities of the agents in conflict
v. Number of conflicting parties
vi. Differences due to possible lack of information
vii. Anticipated or actual conflict
viii. Conflict is over the effect of a particular alternative (as opposed to the alternative itself)
ix. Conflict is over the goals of discussion (what the aim is to achieve)
x. The importance of agreement towards meeting the global solution
xi. Risk of strategy in the current conflict situation

Each of these dimensions promotes or discourages particular conflict resolution strategies. The degree of conflict and the number of issues involved is important in determining whether a compromise strategy should be adopted. If the degree of conflict is small, a compromise alternative may be acceptable to both parties. However, if search is not computationally expensive, the degree of conflict is large, or a compromise solution is not suitable to one or either party (agent flexibility), a search for other solutions is preferable. If the conflict is over a single issue (i.e. commonly known as 'zero sum') then another parties gain is another parties loss and a compromise solution may be the only alternative.

If there are multiple issues in conflict and the agents have different values on the importance of issues, then both parties can benefit from trading off on the issues that have variable value, in other words, making a small sacrifice on one person's ideals that provides a greater benefit to another.

Agent capabilities

Agent capabilities are the major constraint on the type of resolution method adopted. If the agents have different capabilities, which is fundamental in a design environment where many algorithms are utilised, the resolution process should account for this. The ability to distinguish assumptions from facts is important in negotiation. The human conflict resolution process of 'smoothing' brings in additional information to the conflict that enables resolution. A translation of 'smoothing' capability in computational terms is to allow agents to share information, although this obviously depends on their capabilities and knowledge representation strategies.

Anticipated or actual conflict

Actual conflicts need to be dealt with before design can proceed. Anticipated conflicts may not be dealt with the same vigour as actual conflict. It may require an exchange of assumptions or a change in the design route where the anticipated conflict does not occur.

Value of resolving conflict towards final goals

The importance of conflict resolution towards the final goals should be considered in determining a conflict resolution strategy. If the issue is of minor importance towards the global solution and the computational expense of search is an important factor, then the negotiation mechanism may consider resolving the conflict by randomly choosing an alternative favoured by one of the agents. At least one agent will come away satisfied with the solution. Where the degree of conflict is critical, the search for alternative solutions or integrating goals and issues becomes more acceptable.

Progress of negotiation

The characteristics of the conflict resolution strategies available, and the progress of negotiation effects the selection of an appropriate strategy. The progress of negotiation should be recorded and identify:

i. how much computational resource is required/expected from a specific strategy
ii. how much time has already been spent in conflict resolution

Strategies differ on the resource they consume. An expensive strategy is the random generation of alternatives which may be appropriate if all other conflict resolution alternatives have failed [Sycara 89][Lander, Lesser, Connell 89]. This however becomes

unacceptable in an intractable solution space.

Domain specific conflict resolution strategies

An efficient method of conflict resolution may not be one of those described earlier, but one that was formed from knowledge in the domain. It has been identified that a rich knowledge of conflict resolution expertise can exist for particular conflicts [Klein, Lu 89]. This expertise can be elicited by the knowledge engineer at development time and incorporated into the knowledge based system. These domain specific conflict resolution strategies are likely to be more efficient by leading the agents towards a solution that is usually successful in the subject domain. It may be necessary in certain cases for the conflict to be resolved by the user. This can be regarded as an expensive strategy in the sense that the user should only be consulted when the computer has exhausted its reasonable conflict resolution alternatives.

Negotiation is a dynamic process

It is important to recognise that the selection of a conflict resolution strategy is not a binding judgement. If a strategy is proving unfruitful then it is logical to take steps in selecting another conflict resolution strategy. Negotiation should be seen as a dynamic process where the communication of proposals and conflicts is not one-shot [Sycara 89], and the mechanisms used to resolve conflict may change. Due to the dynamic nature of negotiation, the selection of a strategy may depend on the methods that have been tried previously to resolve conflict. Previous strategies will have consumed resources and these may impose constraints on the selection of a resolution strategy.

Risk

The probable risk of a conflict resolution strategy may be accessed before one is chosen. For example, is it wise to adopt a compromise solution when there is a possibility that a better solution is available ? In this case you have to trade of the risk against searching for a solution and not finding one, against the possible benefits gained from finding an improved alternative. A quantification of risk may be possible from knowledge of the bounds of the search space and an idea of the solutions that you will find.

Selecting a strategy is a knowledge based problem

The selection of a strategy is not a simple process and can itself be viewed as a knowledge based problem [Lander, Lesser, Connell 90]. No single strategy is acceptable for all conflicts, and the computational expense, likelihood of success and numerous other factors listed above all complicate the selection of an appropriate strategy. Verifying whether the correct strategy has been selected is complex. If one strategy is quicker at coming up with the same solution than another, the other agent may have taken more time considering the various alternatives at a deeper level of complexity and therefore has a more reliable

solution. Since optimality in design is also elusive, the comparison of solutions from the various resolution techniques also presents a complex problem. The generation of rationale solutions in reasonable time limits is considered to a be the goal of the negotiation mechanism in these complex environments.

5.6. Negotiation

5.6.1 Negotiation as a form of conflict resolution

Negotiation has been identified as a form of conflict resolution in the sense that the roots of conflict are examined and rectified during negotiation [Pruitt 81]. It is a cooperative strategy and appropriate for engineering environments where neither party can achieve their goals unilaterally.

The scope of negotiation however is much greater than those conflict resolution strategies discussed previously. The conflict has to be identified, the problem defined, and appropriate techniques applied to resolve the conflict. This is an iterative process that may require the application of a number of more specific resolution techniques (compromise, smoothing etc) in order to resolve the conflict.

The process of negotiation is similar to the conflict resolution procedure defined earlier. However, *mediators* or *arbitrators* can also be found in the negotiation process. Mediators work with the parties in conflict in helping them to reach agreement. Arbitrators on the other hand listen to the arguments and produces a set of binding recommendations [Pruitt 81].

Variable value

Since negotiation is a cooperative strategy, it is important to ensure that the other parties in conflict receive as much benefit from a small sacrifice by one party, thereby improving the optimality of the overall design. In other circumstances, there may have to be a trade off in values in order to satisfy other requirements (*sacrifice*). It is therefore important to identify the attributes or states that have *variable value*. In other words, identify attributes that have little value to one party, but have considerable value to others'.

Psychological factors

In human workgroups there are many psychological factors that effect the negotiation process. The social status of other participants, the need to preserve self esteem, and the degree of trust between participants [Klein, Lu 89] reflect different attitudes towards methods of negotiation. Obviously computational forms of negotiation will not suffer from these psychological factors. It is important to realise however that the users of a knowledge based concurrent engineering environment will be required to judge and

respond to the system and therefore these factors are not eradicated through such computational support.

In DAI research, negotiation is often proposed as a conflict resolution and information exchange scheme [Bond, Gasser 88]. In a computational environment a language and protocol must be defined that enables the different varieties of negotiation and enable flexible coordination. Computational methods can benefit from understanding the successful attitudes and behaviour of human workgroups towards reaching agreement.

5.6.2 A knowledge based model of negotiation

Figure 12 depicts an overview of a knowledge based negotiation strategy under research. Initially the conflict is identified and a map of the problem space is derived. The map of the problem domain provides the basis on which conflict resolution proceeds [Pruitt 81], and identifies a number of possible causes of conflict.

The agents in conflict, and those that were involved in decisions indirectly related to the conflict then come together to determine the root cause of the conflict. They will have to agree on the root cause from which conflict resolution will proceed. Agreement on the root cause of the problem is itself a conflict resolution process. If further analysis of the cause of the problem does not yield results (i.e. unsuccessful) then other possible causes for the conflict will be analysed. Historic records are maintained on the roots of conflict analysed in order to avoid selection of a root cause that has already been analysed.

The cause of the problem is then analysed to determine the attributes necessary to select a conflict resolution strategy. The attributes cover capabilities of the agents in conflict, number of agents in conflict, number of issues in conflict (e.g. safety, cost, maintenance), number of possible alternatives, agent flexibility etc. The attributes are then taken -together with knowledge of the strengths of resolution strategies -and an appropriate conflict resolution strategy is selected. If previous attempts to resolve the conflict have been unsuccessful then the history of conflict resolution will be consulted. This history documents the time spent in conflict resolution, the computational resource available, and failed resolution strategies. Particular constraints on the selection of a resolution strategy (e.g. time, computational resource) may be imposed upon analysis of this history. This research aims to establish a relationship between these attributes and the appropriate conflict resolution strategies.

The chosen conflict resolution strategy is then applied. Boundless resolution strategies are monitored and terminated if they exceed their available resource or time limit. If the strategy is successful (i.e. the conflict is resolved) then the conflict and resolution method are recorded in order to improve future system performance. If the strategy is unsuccessful, then the conflict resolution history is updated and the roots of conflict re-analysed. If a solution does not seem tenable by applying conflict resolution techniques

to the current root cause, then other causes of conflict will be analysed. If the current root cause has not received rigorous analysis, and other conflict resolution strategies are available that appear appropriate for the conflict situation, then these strategies are applied to the same root cause.

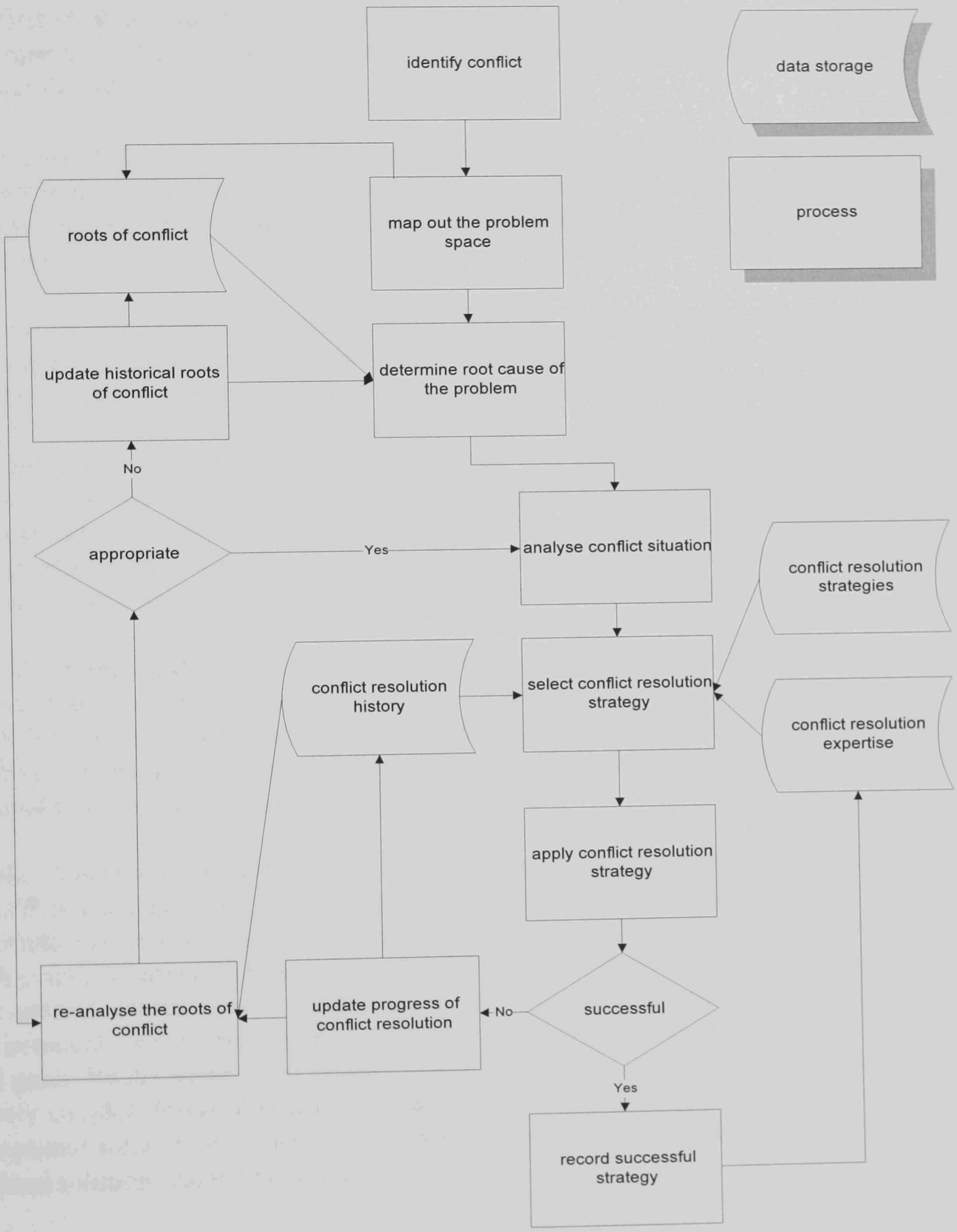


Figure 12 The Knowledge Based View of the Negotiation Process

The model shows the negotiation process as a highly iterative strategy, possibly covering an analysis of many root causes and many conflict resolution strategies.

5.7. The results of conflict resolution

The first obvious result that is required from a conflict resolution process is that there is no longer a conflict. Assuming that there is a solution, it is then appropriate to ask how optimal the solution is.

The scope of the word 'optimal' is important to clarify. The design of an engineering artifact requires the resolution of many conflicts. The result of the conflict resolution process could be either optimal with respect to the conflict, or optimal with respect to the complete design.

Optimising the result of a single conflict may be a simple task. If there is some non-subjective global evaluation function or weighting criteria, and the values in conflict can be directly translated into these terms, then the optimal result can be obtained by simple selection of the most highly weighted option. The task however becomes more complicated if the values cannot be translated into goals, the weighting criteria are subjective, or the options have considerable knock-on effects to later parts of the design process that have to be considered for an optimal solution to be found. Additionally, the design of a global evaluation function itself can be a considerable knowledge elicitation task and looses some of the advantages gained in an agent based approach.

In the normal sequential design process, where each individual or design group in turn applies their own criteria and judgement to the decision process, an optimal solution is unlikely to be found with respect to the end product. Optimal solutions in the early stages of design without regard to global concerns, are likely to restrict and impinge on the optimality of further decisions throughout the design.

In the concurrent engineering environment, the issues and goals of the design are considered at each stage throughout the design process. If it is assumed that optimal decisions can be made, and consideration is given to all life cycle goals and concerns at each point of conflict, then the end product is more likely to be optimal. The assumption that optimal decisions can be made however is a big one. In complex engineering design environments where there are many complex decisions and tradeoffs, translating values into goals, the development of global evaluation criteria, and the analysis of the effects of tightly coupled design alternatives in the design, all diminish the possibility of reaching an optimal solution to a final product. Sycara [Sycara 89] noted that for such problems optimal solutions cannot be found.

The resolution of conflicts in the human environment is far from optimal. Individuals see the world through their own particular biases and they can easily distort reality [Gray,

Starke 84]. Where there are many problem dimensions, issues, criteria and alternative solutions it becomes increasingly difficult for humans to adequately comprehend and evaluate the alternatives [Anson, Jelassi 89][Cutkosky, Conru, Lee 94]. In the face of growing complexity, common behaviour is to ignore the complex interdependencies during the early stages of design [Cutkosky, Conru, Lee 94], therefore serialising the process. In this case inferior design decisions do not surface until later in the design process. Concurrent engineering is a step forward in resolving the evaluation of the larger grained decisions early in the design phase, but there are clear benefits by having computational support.

To conclude on the results of conflict, the aim is to achieve a solution whereby the group of design engineers can perceive the solution as the 'best collective solution'. If there is disagreement over the solution, then obviously their exists some conflict left to resolve, or more justification is required to convince the individual that the best collective solution has been found.

5.8. Summary

A conflict is a disagreement between two or more viewpoints on some decision or value proposed in a design. These disagreements are a result of the differing *needs* that lead to incompatible preferences among the alternatives under consideration [Pruitt 81]. The chapter detailed a review the nature of conflict, how conflict is recognised and various conflict resolution mechanisms. The text continually refers to conflict resolution in the human workgroup in order to develop an analogy and therefore promote the development of computational techniques. The chapter explored in some detail each strategy for resolving conflict, both in the computational and human workgroup. The chapter covered the various stages of resolving conflict: agree that a problem exists, understand the problem, identify the root of conflict, generate alternatives, and evaluate and select the most appropriate alternative.

To resolve conflict in CDEX, an appropriate conflict resolution strategy must be selected. The chapter describes the dimensions of the problem domain that exist, and how these dimensions effect the choice of an appropriate conflict resolution strategy. A knowledge based model of negotiation is presented that highlights the important aspects of a negotiation model. This model is presented from a higher level perspective. The next chapter describes how CDEX implements the negotiation model, and defines the measurable factors in the knowledge based domain that are important in the selection of a strategy.

Chapter 6. The Concurrent Design Expert (CDEX)

6.1. Overview

The previous chapters on the design framework and negotiation metaphor have provided the theoretical background and basis on which to support the development of a system. The arguments and design approach to the implementation of the individual parts of the framework have been covered. This chapter aims to pull all these ideas together into a detailed design of a framework to support the requirements identified. Object oriented principles have been applied throughout the development of CDEX and therefore it is appropriate that this chapter discusses the detailed design around these object concepts. Initially, an overview is provided of the three main objects in CDEX, the design object, the proposal, and the refinement object. These are discussed from an operational perspective to shed light onto the operation of the system. A more detailed design can be found in appendix K which includes the object diagrams and state transmission diagrams. The CDEX grammar is described that enables communication to take place within the framework. A more detailed review of the CDEX grammar can be found in appendix L.

In the previous chapter on conflict and negotiation, various conflict resolution strategies were described and the problem of negotiation was described as a knowledge based problem. This chapter goes into the detail behind how conflicts are assessed in the framework, and how the strategy is selected and applied. The implementation of the conflict resolution strategies are described from the technical perspective with regard to manipulation of utility weightings and assessment of applicability to resolving conflict.

6.2. The CDEX Approach

This section provides a description of the CDEX design tool from a general and operational viewpoint. The main objects and object behaviour in the system is described. For more detail on the design of CDEX (object models, definitions etc) see Appendix K.

6.2.1 Design Objects formulated using the OO Paradigm

Two important hierarchies in object oriented design - the 'is-a' and 'is-part-of' relationship hierarchies - map well onto the design methods identified. The 'is-a' hierarchy is used to represent a tree of objects that inherit the properties and behaviour of those objects higher in the tree. This hierarchy maps onto the selection process when viewed top-down. For example, both centrifugal and positive displacement are types of pump which are represented in the 'is-a' hierarchy. Therefore when a pump is required, either a centrifugal,

or positive displacement type pump must be selected. Figure 13 shows the example where a type of buffer can be selected as either a storage buffer or a tankered buffer. The storage buffer can in turn be selected, and either a tank or pressure vessel proposed as a result.

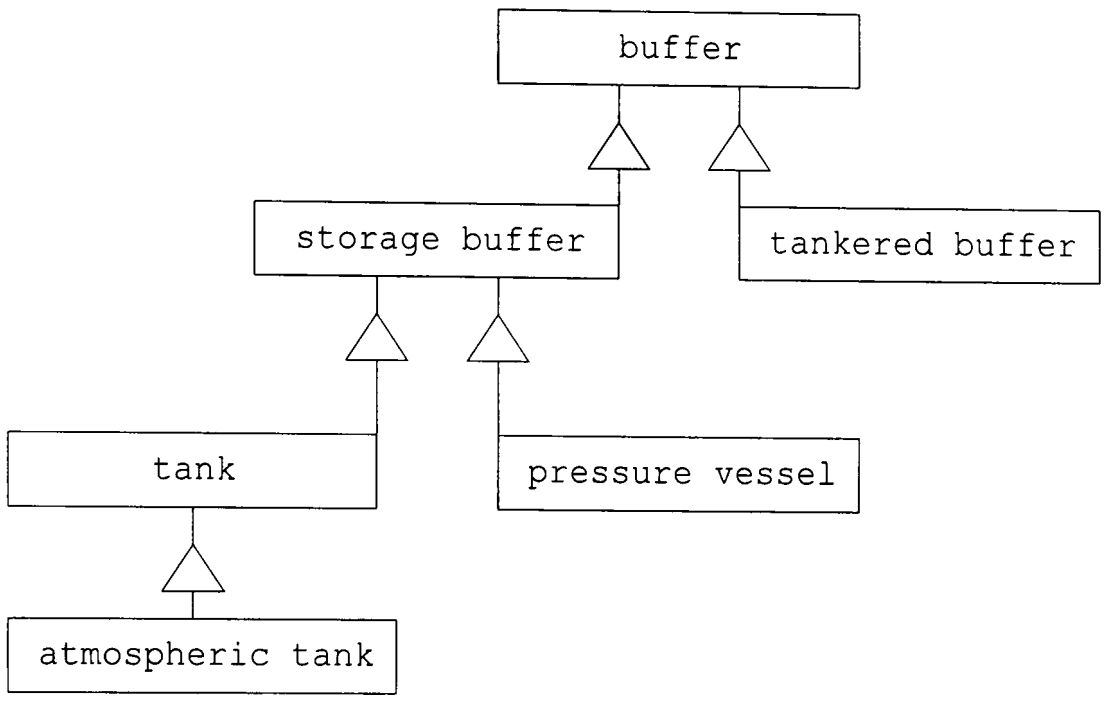


Figure 13. Section of an object hierarchy in the design space.

The 'is-part-of' hierarchy denotes that an object is part of another object. For example, a pump, a heat exchanger, and a pipe are all parts of (is-part-of) a heat transfer system. This hierarchy maps well onto the design synthesis process.

The parametric design function is the assignment of values to attributes of an object. The process is performed from an analysis of the requirements. These requirements are documented in the parent object in the hierarchy, and other objects derived from this parent.

Besides Object Oriented design just being a good mapping onto the three different engineering design processes, there are other benefits. The structure of the object hierarchy is a neat method for organising the knowledge elicitation exercise. In other words, there is a distinct separation of knowledge that is required to synthesise, select, and size a component. This knowledge can again be classified by the object under analysis (e.g. a pump, heat exchanger) and by the general nature of the object: for example, general nature that pertains to all pumps, or more specific knowledge about a particular pump type such as centrifugal. A second advantage is the availability of object databases, rule based systems, and development languages.

6.2.2 The three basic elements: Design Object, Proposal, Refinement

The functionality of CDEX is essentially encoded in just three objects: the Design Object,

the Proposal Object, and the Refinement Object. These three objects have the ability to cooperate and enable cooperative design to occur between agent systems.

The Design Object

The importance has previously been highlighted of a standard for representing engineering design information and the necessity of a standard in a distributed system. In CDEX their is a central data model that represents all the different types of engineering equipment in an object hierarchy (see Figure 13). The instances of these objects are all held in the design space and conform to the standard data model.

The hierarchy depicted in Figure 13 is a small section of a much larger design space. Design progresses from an initial requirement for a 'buffer' (something to hold product or feed) though to something more detailed such as a pressure vessel. What is not depicted on the diagram is the base object from which all of the equipment/design objects inherit. This base object is the Design Object and has specific behaviour.

The basic objective of the design object is to maintain the dependencies with other design objects within the design context. If design has proceeded on the information contained in the design object and the information is either changed or deleted, then the other dependent design objects must also be retracted from the design space. A dependency link with the parent object (the more general objective/requirement) is also maintained. This dependency is important in the selection process as a general attribute that is modified in a more specific design object, must inform the parent design object which may in turn retract other dependent design objects in the design space.

The Proposal

The basis mechanism for agents to communicate design information and evaluations is the proposal. Conflicts, evaluations and designs are all grouped under the banner 'proposal' because of their shared characteristics. All proposals - both conflicts and designs - are based on the requirements and design defined elsewhere in the design space and the dependencies between the proposal and other design information has to be maintained. If some part or attribute of the design is modified or deleted, then all proposals based on this information must be retracted.

The basic proposal records the agent who put forward the proposal, the confidence and preference that the agent has on his proposal, and the rationale behind the proposal. The rationale is a text description that is useful in an environment where the engineer has presented the proposal and wishes to record his rationale behind the particular proposal. The rationale is not required for processing in the computational framework.

Proposals are classified into three different types: Option proposals (which include parametised, synthesised, and selected designs), conflict proposals, and evaluation

proposals. Evaluation proposals essentially record the information highlighted above with the basic elements of a proposal.

The Option Proposal records the design objects which an agent considers should be part of the design. References to external evaluations and conflicts are also maintained which ease the assessment of a proposal - as all views on that particular part of the design are recorded in one instance.

Conflicts are presented for Option proposals when an agent disagrees with a design solution. Conflict proposals record information that can help the conflict resolution process as well as highlighting the severity of the conflict. Such information includes the type of constraint (hard/soft), a type of deviation together with limit values (e.g, high temperature - greater than 90 degrees C), or a problem keyword. A set of generic problem keywords can be developed for different design objects depending on its function. The HAZOP style guidewords [Kletz 85] and parameters are being applied to lines (e.g, high flow, no flow), and vessels (eg, high level), and more prescriptive ones for components (e.g. reverse pump flow). These keywords can be made available in the framework for agents to base specific conflict resolution strategies.

A conflict is treated as a proposal as it is a viewpoint presented by an agent (albeit regarding another proposal) that can be dependent on other information in the design, including the plant topology. Conflicts can be removed from consideration given that the design information on which the conflict was accessed is changed.

The Refinement

Each design object has three dependent refinement objects which manage the detailed design process of the specific object in question. The three refinement objects represent the parametric design, synthesised design, and design selection of the design object to which they are attached. A refinement will record proposals put forward by the agents in the environment (including problems identified and other agents' views) , and apply a negotiation process to determine which proposal should be accepted.

For example, a storage buffer will have a refinement object which looks after the selection process. This refinement object will collect proposals concerning the selection, these proposals may contain such things like a pressure vessel or a tank. The refinement object will then collect any reviews or conflicts that have been identified with the different proposals and negotiate the best proposal to work with. This refinement process is not 'one-shot'. Conflicts and other alternatives may be proposed at a later design stage which will mean a review of the proposals.

The refinement object is responsible for requesting the services of the negotiation mechanism. These services are requested in the following circumstances:

- the last proposal for a particular refinement has been received - this is known because each agent registers his interests/capabilities (see *Agent* section)
- a proposal that was previously accepted is deleted either due to a design change or request of the user.
- a conflict or review of a design object is raised after the proposal has already been accepted.
- a new proposal arrives when the design has already been accepted. The new proposal will have to be reviewed as it may be an improvement on the currently accepted proposal.

An overview of how the negotiation mechanism resolves conflict is covered in a later section.

The refinement objects involved in the selection and synthesis processes also keep track of dependencies to other proposals. When a particular refinement has been accepted by the negotiation mechanism, it becomes available to other parts of the design process to make decisions based on the topology of the plant. Any proposal that is based on information pertinent to the topology - for example: "is there a valve on the pump inlet?" - must be recorded as a dependent of the refinement. Therefore if the refinement design object is deleted, or the accepted proposal in the refinement is modified, then the proposals based on that refinement (including identified conflicts) must be re-accessed.

6.2.3 The rationale for both 'Conflict' and 'Evaluation' proposals

"Whether we want to, and whether we can"

Two issues are involved in the CDEX negotiation process: the **physical** and the **value**. In other words, whether we would like something to be a certain way, or whether it is possible for something to be a certain way. There are not too many problems with the physical, as the agents do not generally put forward solutions that are not possible from the structural viewpoint which is what one is mainly considering in the test case. Additionally there are not many things that physically cannot be achieved structurally either. For example, an agent could put forward a proposal for a tanker to unload directly into the distillation column. If it needs to be filled from the top, it is possible to imagine the tanker driving up a ramp to the top of the still, and pouring its contents into the top. This is not a physical problem, but is value problem, particularly with regard to the cost of building a ramp capable of supporting the tanker.

To sum up, **values** dictate whether an individual has a *need* to do something, and something outside of an individual's control dictates whether it is possible. It is reasonable to assume that the rules that dictate whether something can be achieved is 'outside of any individuals control' as if this was not the case, an individual could direct its behaviour to ensure that it was possible.

The 'Soft' and 'Hard' conflicts

A SOFT conflict is no different from an evaluation in that it presents the quality of the proposal from an agent's perspective. The exception is that the soft conflict includes an identification of the problem with the proposal which therefore provides an indication of why the preference of the proposal is not as high as it might be. This information is useful in the CDEX framework as another agent may have the capabilities to improve on the proposal if he understands that a particular problem exists. The problem identified in a soft conflict will not result in a solution that will not work (such as feeding product from a low pressure source into a high pressure tank) but identify general undesirable characteristics of the proposal (e.g. a lot of dangerous product stored on site).

The preference identified with a conflict is a preference for the solution. The preference may be high if the solution looks good, although a conflict may still exist if the agent does not believe it is achievable. It is therefore possible to have high preferences with a conflict.

A HARD conflict on the other hand denotes something that cannot physically be achieved. A HARD conflict with total confidence indicates that something is definitely not possible, and the agent believes it to be so without doubt and therefore the proposal put forward is not acceptable. When is it the case that there is a *low belief* in a HARD conflict? when one is unsure about a fact (information not available) or when design is taking a route which is considered to be unachievable.

Dealing with the Hard Conflicts

The confidence associated with a proposal is related to the confidence in the design attaining the desired preference. This is the same with conflict proposals, and therefore the confidence associated with these proposals is not the confidence or 'chance' of the hard conflict preventing design from continuing further down the design path. Any lack of confidence associated with a conflict proposal will however be attributed to the potential for the specified conflict to occur. The potential of the conflict occurring will therefore be equal to 1 (i.e. the confidence defined in the conflict proposal).

If there is a total lack of confidence in the approach due to an agent believing an approach to be impossible, then design cannot continue down the proposed route. The question remains of how to include an assessment of a hard conflict, in which the agent does not have complete confidence, into the assessment of the other viewpoints in order to determine an appropriate design route. If two hard conflicts are considered each with the same preference (i.e. both viewpoints consider the benefits in the end result of taking the particular design route equally), with the exception that agent A has confidence of 80% that the conflict will occur, while agent B has a confidence of 50%. When considering which is the better evaluation, one would say agent B's (the confidence of 50%) as it is less likely to hit trouble, and more likely to achieve the design quality denoted by preference. The reason agent B's evaluation is preferred is not because the potential reward is greater (both

agents have assessed the design quality to be the same), but that it is more likely to save 'time' - as it is less likely to hit a problem. 'Time' is therefore the factor to be modified according to the degree of confidence of a hard conflict. When a conflict resolution strategy puts forward an estimate of the time required to resolve the conflict, this time will account for the confidence of the hard conflicts associated with the proposal under review. Some strategies will obviously be less affected by this, for example, the 'generation of new alternatives' will not be affected at all, as potentially a completely new solution will be put forward where the problem may not exist.

When assessing a number of hard conflicts associated with a proposal, none of which have a confidence of 1 (in which case design will not proceed at all), one has to account for an overall effect on time. The problem here is that if two agents believe the design path will result in problems that prevent design from continuing, one cannot be sure that the root cause of the problem they have both identified through conflict proposals is the same. If two agents have identified the same potential root problem, then a pessimistic approach could be adopted which uses the worst confidence associated with either of them, or it could be assumed that both problems are independent, in which case the potential for failure is much greater (the sum of both confidences!). One cannot put this problem down to agents not being able to share knowledge. An agent may have made assessments from the past that particular configurations of design for example just cause problems, without having the exact causal knowledge of why.

A mechanism for resolving this problem could be a standard for representing the cause effect chain with the conflict rather than just the simple keyword as defined at present. For example, NO-FLOW for a pipe could be put down to "POWER SUPPLY(fail) -> PUMP (stop) -> PIPE (no flow)" (where x -> y means x is the cause of y, and the bracketed terms denote the deviation in the equipment item). This would provide the negotiation mechanism with additional information which it could use to determine if the conflict being proposed by an agent is mutually independent of any other conflicts identified with the proposal, may be some sort of quantified risk assessment procedure would be appropriate. Issues here regard whether the potential terms for failure (e.g, NO FLOW - the standard HAZOP terms [Kletz 85]) encompass the complete requirements of the root causes for failure that the agents may consider. This is left as a problem for future research.

CDEX manages the potential for conflict and the issues regarding multiple conflicts in the following manner:

- The time increases as the confidence in the conflict increases in the following manner:

  (1 / (1 - confidence in proposal)) * time expected to resolve conflict

- If there is more than one conflict in the proposal, then the times derived for each conflict are summed.

This gives us the desired behaviour of disliking a design path if both time is important and the confidence in potential conflict is higher, as well as disliking a design path if more than one conflict has been identified.

6.2.4 Agents

Agents act autonomously in CDEX to enable the application of the many different design technologies required to support the lifecycle design issues. The cooperation between agents however requires a standard interface - or 'wrapper' mechanism - to support the structured design process.

On startup agents are required to provide a list of their capabilities to a *control layer*. These capabilities are recorded by design function (i.e. synthesis, selection, or parametric) and design object (e.g. pump, heat transfer system). Agents create proposals by first creating the design object(s) that are part of the proposal. The agent then places the design objects into a proposal, and sends the proposal to the CDEX control layer which ensures the relevant refinement object is informed. The agent can also specify the confidence it has in the proposal if it has the capability. Agents that have indicated their interest in evaluating particular parts of the design will then be informed.

When an agent is requested to evaluate a design, the agent will either present an evaluation, a conflict, or a message to say that it cannot review the design it has received. The negotiation mechanism will only continue with reviewing the proposals for a particular design object when all interested agents have responded. The agent may be acting as a simple wrapper to the external software program (e.g, CAD which performs a 'synthesis' function) and putting these designs forward for review.

An agent's interests are specified using a simple grammar. Section 6.3 depicts the main elements of this grammar.

In the engineering design process each discipline has its own priorities, objectives and goals to satisfy. The same issue applies in a distributed environment, where different agents encapsulate the different experience and priorities of its designer.

CDEX models these priorities in an *objective hierarchy* [Keeney, Raiffa 76]. Each agent may have its own methods for determining the utility of a design, but at the same time must be able to relate these objectives to a predefined standard objective model. This is important in facilitating the negotiation process as there is a need to be able to identify competing goals, and be able to exchange goals to determine if conflict is over the goals of discussion, or the effects of selecting a particular design option. An example set of higher level objectives in CDEX that are shared by all the design disciplines are: cost effective; meets duty requirements (rated and alternate); maintainable; operable; and meets the physical constraints.

6.2.5 OO Framework supports distributed design

The development of CDEX using object oriented principles has had an important benefit with regard to easing the distribution of processing. As described in section 6.2. the main functionality behind CDEX is developed in three main objects. the proposal, the design object, and the refinement (more detailed design). A fourth object, the 'Negotiation' object - manages negotiation and conflict resolution when needed for a particular refinement. There is therefore a negotiation object for each refinement if conflict is identified. Using this OO principle enables us to distribute the processing for resolving conflict for the many different design refinements over any number of machines. Potentially the negotiations for each design refinement can be executing on different machines at the same time. The traditional AI search algorithms fit well into the OO paradigm. Consider the following standard AI search algorithm for traversing a tree and performing a function 'doSomething' on each element:

```
function expand (node)
{
 doSomething (node);
 list-of-children = generate-children-of (node);
 for each child in list-of-children
        expand (child);
}
function doSomething (node)
{....
}
```

An object oriented version of this could be where each node is an object itself, and each node is responsible for controlling its own children:

```
class node
 method doSomething ()
 {....
 }
 method expand ()
 {
        list-of-children = generate-children-of ?self
        list-of-children = generate-children-of (?self);
        for each ?child in list-of-children
              send ?child expand
        send ?self doSomething
 }
```

In this definition ?self refers to the instance of the class itself. Messages are sent to objects with the primitive 'send' rather than by explicitly calling a function. This approach is conducive to distribution as the object itself does not have to reside on the same computer. The address for an object could be the combination of machine and object location in memory for example. The functional approach has a single thread of control, whereas the OO approach has multiple threads. The functional approach above has the property that each node is visited in an explicit order. In the OO approach this ordering is not explicit. CDEX does not require the ordering to be explicit, and due to the need for distributed processing to provide power, flexibility, and maintenance of diverse knowledge bases and design tools, the OO approach was appropriate.

6.3. CDEX Language

6.3.1 General overview

The agents cooperate through the CDEX framework through a simple language that enables them to put forward and review proposals. The grammar is purposefully simplistic to reduce the burden on agents who are expected to translate the requirements specified through the grammar into a format that can be purposefully analysed.

The three main elements of the grammar are PROPOSE, DESIGN and EVALUATE. PROPOSE enables an agent to put forward design requests, evaluations of other proposals, conflicts with other proposals, design refinements (selection, parametric, and synthesis), and alternatives. DESIGN is a request to the agent from the framework to progress a particular design path. The DESIGN requests obviously includes the definition of the requirement (a design object) and type of design required - either parametric, selection or synthesis. The EVALUATE is also a request from the framework. EVALUATE is sent to an agent as a request to evaluate a proposal from the agents individual perspective. EVALUATE requests are triggered by the framework when proposals are put forward by other agents. These DESIGN and EVALUATE requests are only sent to an agent if the agent has noted his particular interest in the appropriate style of design (selection, synthesis, parametric) and design object.

6.3.2 CDEX Grammar

The following section describes the CDEX grammar that agents use to communicate within the CDEX framework. A description of the grammar in BNF notation is provided, together with an overview of the general purpose of the key terms. A more detailed overview of the grammar which may answer more specific questions is supplied in appendix L.

Dictionary

The language definition conforms to the BNF notation.

<agent>	The name of the agent.
<design-method>	The design method, either 'selection', 'parametric', or 'synthesis'. It can be 'null' (by default) if the proposal is a request for the design process to proceed, e.g, the user wants a design for a heat transfer system from an agent (or user).
<req>	A textual description of a requirement. Must hold a text description that can be given to an engineering user - a reason why for example an alternative solution must be provided. This is to allow normal human interaction with the system.
<reason>	The reason for a particular conflict. A reason can be either formal or informal. Informal: a description (textual) of why a conflict existed Formal: a notation enabling an agent to understand and apply a relevant conflict resolution strategy.
<obj> <objReq>	A reference to a design object.
<importance>	The importance ascribed to particular problem. It can denote whether a conflict is due to a hard or soft constraint violation, and identify a particular dislike to a solution.

PROPOSE

Syntax: **PROPOSE** <proposalObject> [[**for** <desObj>] | [**for** <proposal>]] **from** <agent> [**as** [**initial** | **alternative** <altern-to-proposal>]]

Parameters:

<proposalObject>	[<parametised proposal> | <synthesised proposal> | <selected proposal> | <conflict proposal> | <evaluation proposal>]

<desObj>	The object to which the proposal is associated with (e.g. A design such as a refinement).
<proposal>	The proposal with which the proposal should be associated. This is specified for evaluations and conflicts which are associated with option proposals rather than particular objects.
<agent>	agent presenting the proposal
[initial | alternative]	indicates whether the proposal was a result of a design request (initial) or an alternative (after a request to provide an alternative). The specification of initial or alternative only makes sense for an OPTION proposal.
<altern-to-proposal>	A reference to a proposal. If the proposal is an alternative to one previously presented, then the alternative proposal must be specified.

General notes:

Proposals are the method of communicating designs and design understanding. Agents present different proposals depending on the type of design (synthesis, selection, parametric) and whether they are supplying an evaluation or identifying a conflict regarding a design proposal. Proposals have important characteristics in that they record dependencies to other design proposals and design objects. The maintenance of these dependencies is important for if design is modified due to some circumstance, the proposals based on that modified design information must be retracted and the situation reviewed. If the <desObj> is not specified, it is assumed that it is new proposal containing a system that requires design.

DEVELOP

Syntax: DEVELOP <desObj>

This is a request to a design object to design itself. When the negotiation mechanism wishes to inform an object to continue its design for review purposes, or a proposal has been accepted, then the objects must be sized, synthesised, and possibly go through a selection process. DEVELOP is a keyword that informs an object that these processes can begin.

DESIGN

Syntax: DESIGN <method> <desObj>

The design keyword is a request for the agents to design an object, i.e. to present their proposals.

The DESIGN request is issued when the design space has received all the appropriate evaluations that were requested for a proposed design and one has been selected and now requires further design.

The method is one of PARAMETRIC or SELECT_OR_SYNTHESISE. If parametric, then only objects that can perform a parametric design process can contribute.

ALTERNATIVE

Syntax: ALTERNATIVE <agent> required for <method> <obj> [because {<conflict>}] [using goals <utility-vector>]

A request from the negotiation mechanism to a specific agent to produce an alternative.

The agent responds to the request for providing an alternative by presenting a proposal. The agents previous proposal does not need to be retracted. The proposal will be evaluated (as with other proposals) and re-considered again as part of a group with all other proposals.

An alternative may be required because of a set of conflicts. The agent, if it has the capability can ensure that the next proposal generated does not disregard the conflicts that were identified with the initial proposal. These conflicts are parcelled as part of the request to provide an alternative ('because {<conflict>}').

If the agent can exchange goals, then he may be requested in a situation to apply certain goal criteria <utility-vector> in proposing his solution.

EVALUATE

Syntax: EVALUATE <proposal> for design of <desObj> [using goals <utility-vector>]

After a proposal is received it must be evaluated. An EVALUATION request is sent from the negotiation mechanism to the agent. The agent responds with a proposal of type evaluation. The design method is indicated by the type of proposal. An EVALUATION is not sent to the agent that put forward the proposal.

The negotiation mechanism may request an agent to perform an analysis of a design proposal using certain goal criteria (for example - where the goals have been compromised). This new goal criteria to apply is specified in <utility-vector>.

REJECT

Syntax: REJECT <proposal>

An ACCEPTED message is sent from the negotiation space to the agent when a proposal is accepted. At the same time a REJECT message is sent to the appropriate agents for the remaining proposals. The reject message is sent to those agents that presented proposals and those that presented evaluations/conflicts for the proposal.

This is useful for management purposes if a system keeps track of its proposals and wishes to maintain a list of which ones have been accepted and which ones rejected. Note that a rejected proposal may become accepted at a later date due to the previously selected proposals causing design problems. Therefore if an agent keeps track of which proposals are reject and accepted, it would be wise not to delete a rejected proposal just because it had been rejected.

ACCEPTED

Syntax: ACCEPTED <proposal>

When a proposal is accepted, all agents that were involved in evaluating, proposing, and presented conflicts regarding the proposal will be informed. This is for management purposes if the agent wishes to keep track of the areas of design in which it was involved. The ACCEPTED command is sent from the negotiation mechanism to the agent when a proposal has been accepted. However, it is possible that problems later in the design may cause the proposal to become rejected and therefore the agent may account for this.

FORCE_DELETE

Sytnax: FORCE_DELETE [<proposal> | <desObj>]

The DELETE message is sent from an agent to the negotiation layer and is considered to be a 'remove from consideration' command. Essentially it is the reverse of the proposal command with no requirement. If the object has no parent, then there are no issues, and the dependencies will be deleted. If the object however has a parent then the parent must be informed and the objects proposal deleted as well. Dependencies for the object must be removed.

NEGOTIATE

Syntax: NEGOTIATE <refinement> because <reason>

This command is asserted by the REFINEMENT object when all the proposals become 'ready to review'. This fact indicates work for the negotiation mechanism which has to review the proposals and conflicts.

Parameters:

<reason> The reason why negotiation should procede. The <reason> can be either:

'null'	for no reason, just negotiate a best proposal
'POTENTIAL_ERROR'	if there is a potential design error, i.e. accepted proposal has new conflict or review
'POTENTIAL_IMPROVEMENT'	if there is a potential design improvement, i.e. a new proposal has arrived in the refinement, or a new evaluation has arrived for another proposal in the refinement that was not accepted.

Syntax: NEGOTIATE <refinement> for <altern-proposal-object> as alternative to <old-proposal-object>

This message is used when alternative design proposals are received which have been requested by the negotiation mechanism.

INTEREST

Syntax: *form1* INTEREST <agent> EVALUATE <design method> <object type>

form2 INTEREST <agent> DESIGN <design method> <object type>

Parameters:

<design method> either PARAMETRIC, SELECTION, or SYNTHESIS

form1 An <agent> asserts an interest in evaluating the particular <design method> design of <object type>. An agent is informed of events based on his interests. The objects he is informed about concern the <object type> he has specified, and any of the superclasses of this <object type>.

form2 When a DESIGN request is issued, only those agents interested in performing a certain design function <design-method> for the specified object type <object type> are informed.

Note a DESIGN may be requested for SELECT_OR_SYNTHESISE, and two INTERESTS may have been indicated by an agent for SELECT and one or SYNTHESISE.

UNABLE_TO_PROPOSE

Syntax: UNABLE_TO_PROPOSE <agent> <design method> for <desObj> [as [initial | alternative]]

When an agent <agent> has specified an interest in presenting a proposal, and cannot do so (may be because certain structural information is not available) then the agent responds with UNABLE_TO_PROPOSE the <design method> design of <desObj>.

'INITIAL' or 'ALTERNATIVE' is required so that it is known whether the agent has responded. If INITIAL, there is a need to inform the WAIT objects, otherwise not. If an alternative is not available to be proposed, the NEGOTIATION object will be triggered to analyse the situation, and potentially request another agent to put forward a proposal.

UNABLE_TO_EVALUATE

Syntax: UNABLE_TO_EVALUATE <agent> <proposal>

When an <agent> has been requested to evaluate a proposal <proposal>, one will expect a response. In this case the agent can not evaluate a solution and therefore informs the negotiation mechanism that it is unable to evaluate a solution. When all proposals have been received, then the option proposal is informed that the review is complete (review-complete).

6.4. Approach to resolving conflict

6.4.1 Negotiation

In order to resolve the conflict inherent in design, CDEX incorporated an approach to resolving conflict- the 'Negotiation' metaphor. Utility theory is documented as providing a good mechanism for representing engineering preferences which the negotiation mechanism requires in order to maintain the general best interests of all the engineering agents in the design process. When assessing a design problem or solution, an agent will measure variables in the problem domain and relate these to how well their objectives (preferences) are being met. Using the utility theory approach, given the agents preferences, and the objective measurements taken from the problem domain, a single value can be calculated which represents how good the solution is from that agents perspective. The negotiation mechanism can use these values determined by the different engineering agents to identify the solutions most preferred. If the highest weightings are for the same design approach then no conflict exists. However, if the agents do not agree on the best approach, utility theory itself does not help us in deciding which proposal is in the best interests of all the engineering agents involved. A decision based purely on the highest weighted solution would discount the viewpoints of the other agents who presented a viewpoint - even though their view of the solution may be extremely poor. In engineering design there are a variety of disciplines each with their own viewpoint, disregarding the viewpoint of any individual is to the peril of the project, and ultimately the design is only good as the collective view of all the engineers involved (including those who operate the plant). Consider the scenario just described, where agent A considers solution 1 as highly effective, and agent B considers solution 1 as poor. In this scenario is another solution - solution 2 - that is highly effective from both agent A's and agent B's perspective, although agent A considers solution 2 to be slightly less effective than solution 1. In this case it would be reasonable to assume that solution 2 should be adopted as both consider it appropriate. What would the mechanism be to determine that solution 2 was the best to go with? In this case one could apply a consensus strategy, i.e. choose the proposal which most of the agents rank highly. This would resolve the conflict in this particular case, but their are many other issues to consider such as the potential for hard conflicts to exist (where an agent considers a proposed solution impossible), where all agents do not like the approach, where not all the agents rank a solution highly etc.

The research concentrated on identifying methods to resolve the multitude of different conflicts that could occur, given that one can represent the engineering preferences with utility vectors, and that each agent could identify how an alternative satisfies his own personal ideals. Through looking at the mechanisms at how human groups resolve conflict (compromise, goal integration, drop the least important issues, etc), it was possible to take these approaches and develop mechanisms that model these approaches through manipulation of the utility vectors. Although the approach is not as scientific as the kinds of detailed analysis that can be performed with utility theory, a single utility vector is not being considered (only one viewpoint) which is where utility theory is very effective as a method of identifying the best approach. By modelling the human approaches for conflict resolution, it was hoped that the solutions selected as a result of automated conflict resolution would be the same as those solutions selected in a human conflict resolution scenario, thereby enabling the computed solutions to be more acceptable.

6.4.2 Determination of conflict

A conflict is a disagreement between two or more viewpoints on some decision or value proposed in the design [Pruitt 81]. Conflict is inherent in an environment where expertise is distributed and this conflict has to be resolved. Negotiation in the CDEX environment is a meta-conflict resolution strategy in that it operates at a higher level to the conflict resolution strategy. The negotiation strategy analyses the conflict, selects and applies an appropriate CR strategy, and monitors its performance.

The first aim of negotiation is to remove conflicts classified as hard constraints. This enables a solution to be developed that will work, albeit probably not to everyones ideals. Assuming that a solution exists, it is then appropriate to determine how optimal the solution is. The results of a conflict resolution process could be either optimal with respect to the conflict, or optimal with respect to the complete design [Harrington, Soltan, Forskitt 96]. If there is some non-subjective weighting criteria used to access a particular conflict, then an optimal solution with respect to the conflict can be obtained by simple selection of the most highly weighted option. In complex engineering design problems however, there are many complex decisions and tradeoffs, and the analysis of tightly coupled design alternatives all diminish the possibility of reaching an optimal solution to a final product. Sycara [Sycara 89] noted that for such problems optimal solutions cannot be found.

After the initial design request is complete and agents have put forward their views on the proposals, a check is made to determine if a conflict exists. This check consists of ensuring that the agents agree on the best proposal, and assuming that they do agree, ensuring that no hard conflicts exist for that best proposal.

If conflict is identified, and a conflict resolution strategy is applied to resolve the problem, then there are additional issues to account for when identifying conflict. For example, if two agents disagree on the best solution, and they are both requested to apply a compromise utility vector in their analysis, the result will be a compromised solution

which is unlikely to be as good as the solutions that they previously put forward. In this case, if one applies the first rules stated above for determining conflict, one will still find that conflict exists, for when the best proposals are determined, they are unlikely to be the solutions developed by compromise. Therefore, by applying the compromise resolution strategy, the agents are already accepting a particular loss of value in the design and therefore will not be in conflict if those losses occur. Agents can always disagree with the compromise approach if the degree of compromise is thought too great by supplying a poor weighting for the solution or by raising a conflict.

The expectations on the results after a conflict resolution strategy has been applied are different if a compromise strategy has been applied. If the expectations of applying a strategy are met after the application of a conflict resolution strategy then conflict does no longer exist. Conflict is therefore not determined by the agents disagreeing on the best proposal, but that one can provide a solution that meets the minimum expectations of all the agents involved with no hard conflicts.

6.4.3 Conflict resolution strategies and strategy selection

There are many conflict resolution strategies that can be applied in different situations depending on the context of the conflict. If the parties are close to resolving conflict then a compromise solution may be appropriate. If however the goal of discussion is critical, or the agents involved in the discussion cannot relax their constraints, then a compromise solution is not appropriate. The following is a list of strategies adopted within the CDEX framework: (for a more complete description of each of these strategies please refer to section 5.4)

- Generation of alternatives
- Compromise
- Abandonment of less important goals
- Integrative
- Smoothing
- Consensus
- Majority rule
- Domain specific

Conflict resolution strategies that have been identified are effective in different contexts. The attributes that are important in determining the context of the conflict are specified in Figure 14 (Conflict Classification Attributes). These variables however cannot be determined directly from an analysis of the conflict situation. In the framework a set of attributes (framework attributes) can be determined which can then be reviewed to determine the values of the conflict classification attributes shown. From analysis of the Conflict classification attributes, a conflict resolution strategy can be selected and applied.

The framework attributes are shown in the table and are mostly provided by the agent. The attribute 'average time of CR strategy' is maintained by the negotiation mechanism and is

	how close parties are to resolving conflict	no. of issues involved	flexibility on issues	agent capabilities	actual or anticipated conflict	conflict over the effect of a choice or goals	imporance of reaching agreement	risk of alternative strategies
Better alternatives available	U							
Preference of choice	U							
Goals being applied		U				U	U	
Hard/Soft constraint			U					
Prominence			U				U	
Agent capabilities				U				
Confidence					U			
Average time of CR strategy								U

Framework attributes ***Conflict classification attributes***

Figure 14. Framework attribute mapping to conflict classification attribute.

the average time particular strategies will resolve conflict. The variables are self explanatory from the previous discussion with the exception of 'prominence'. If an agent can provide a number of alternatives and these alternatives are rank ordered in terms of preference, the agent is more likely to accept requests for other alternatives if the difference in preference between his suggested choice and the next-best choice is not so great. It is therefore a variable that can be used by the negotiation mechanism for determining an appropriate strategy.

CDEX has adopted the approach in CEF by analysing the context of conflict (the capabilities of the agents in conflict) and used some of these variables - notably 'promience' and 'goals being applied' - in the selection of an appropriate conflict resolution strategy. This strategy is different from those that do not depend on the context of conflict, such as the approaches by Sycara [Sycara 89] and Klein [Klein, Lu 89].

Figure 15. shows the relationship between the conflict classification attributes (determined from framework attributes) and the conflict resolution mechanisms supported in the CDEX framework. Only the attributes that support the particular strengths of the conflict resolution strategies are highlighted.

The implementation of the mechanism described above for selecting a conflict resolution strategy is also performed using utility theory. Each strategy makes an assessment of the parameters apparent in the conflict situation (see section 6.4.7- Functions in negotiation) and uses the parameters to develop an assessment of how well suited the conflict resolution strategy is towards resolving the conflict. The parameters determined by each of the strategies is represented as a utility vector with four factors: potential to succeed, preference, confidence, and assessment of time. The preference and confidence are related to the expected design solution that would be attained by selecting the strategy, these are normally averaged assessments of the proposals put forward. These factors are important as the negotiation mechanism should not just pick the strategy best at resolving a conflict, as the best strategy for resolving conflict may be resolving the conflict for a proposal that ultimately no one really likes anyway. The assessment of time is a fuzzy measure of time required for the strategy to be applied (e.g. single shot, iterative, random and potentially no conclusion). This factor is obviously important if the engineering user wants a quick assessment, or whether he minds at all as long as the solution is the best available. 'Potential to succeed' is the important factor from the viewpoint of the conflict resolution strategy in the particular design context. This weighting identifies how appropriate the strategy is for resolving the conflict. For example, if the parties are close to resolving conflict, and all parties are flexible on the issues, then the compromise resolution strategy will have a high value for 'potential to proceed'.

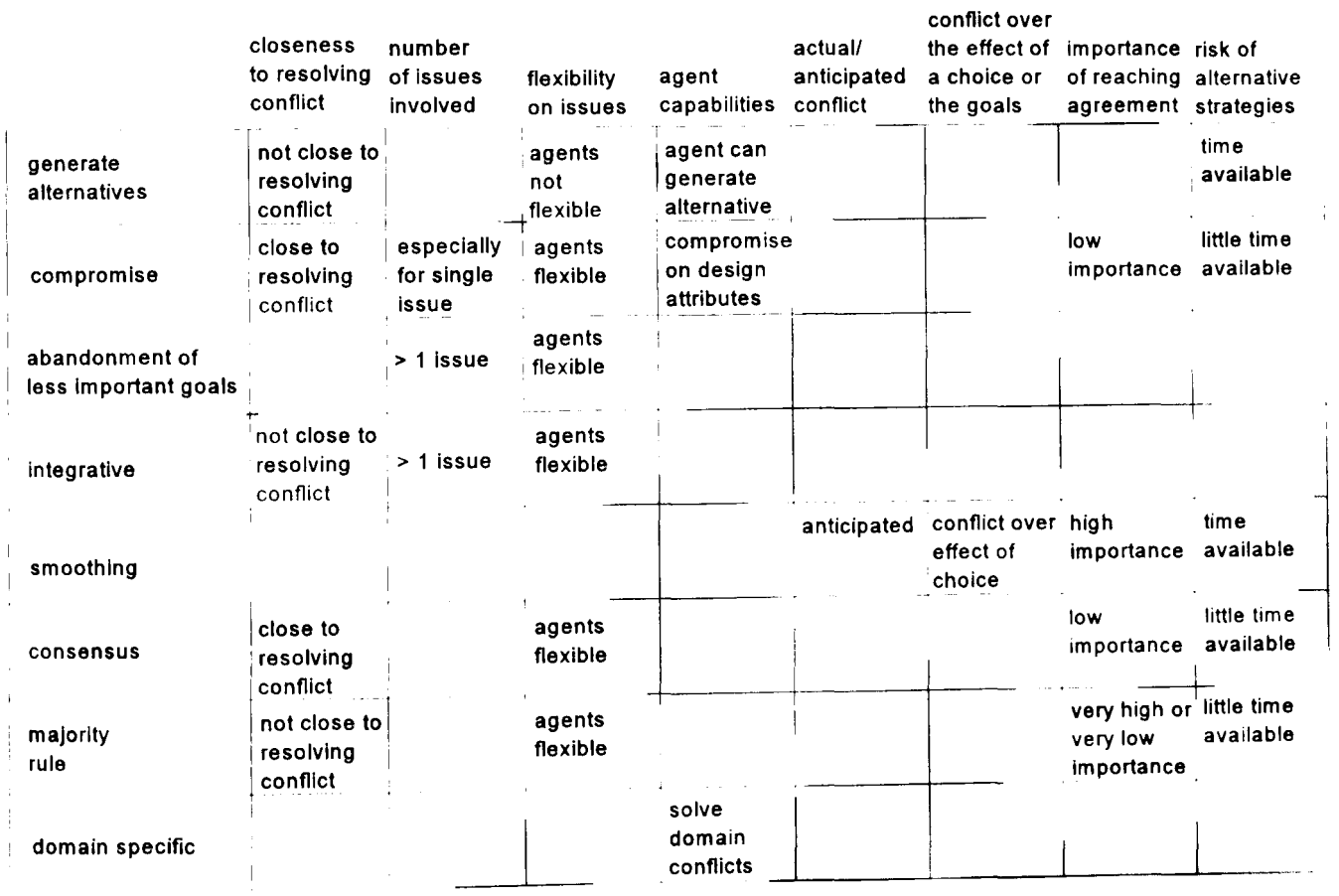


Figure 15. The relationship between the conflict classification attributes and the conflict resolution mechanisms

Depending on what type of design effort the engineering user requires from the system, he can change the importance of each of these factors for a particular design assessment. For example, if the user required a good solution without too much consideration to time, the utility vector used to rank the competing conflict resolution strategies would be something like:

(confidence 0.2)
(potential-to-succeed 0.1)
(preference 0.6)
(time 0.1)

where preference has a very high rating and therefore high preferences have a bigger impact on selecting a design path for assessment. If however the engineer wants a quick assessment of the design problem then the following utility vector will be applied in ranking the different alternatives:

(confidence 0.3)
(potential-to-succeed 0.3)
(preference 0.1)
(time 0.3)

The obvious weighting for the above case is time, which necessarily requires an impact on the selection of a strategy as it is the element we are trying to improve. The other important factors with relation to time are confidence and potential to succeed. A higher confidence will help in traversing design paths that are less likely to fail, and a higher potential to proceed will select a conflict resolution strategy that is less likely to fail.

6.4.4 The 'Concurrent Engineering' Factor

The 'Concurrent Engineering' factor is indicative of the quality of design that would be generated as a result of the framework being applied to a design problem. It is called the 'concurrent engineering' factor because of its characteristics. As design progresses, time is spent, and the assessment of the final quality of the design is likely to change. If one assumes that the best design is required from the global perspective and the design is not turning out like our initial expectations, one would expect to backtrack and proceed down another design path. From a review of the detailed design (Appendix K) the reader will notice that at each stage a strategy is selected by the negotiation object, permission is requested to proceed from the parent negotiation object if one exists. When permission is requested to proceed, the parent is informed of how good the solution is from the global perspective. This evaluation replaces the previous expectations for the design path that the agents had previously derived as the design is present in more detail and therefore the agents are more informed. The parent checks that the design approach is still the most appropriate and agrees that design should proceed if this is the case. If the design path

does not look as fruitful as previous assessments, then permission may be denied and design proceed down another path. Obviously this approach could be very expensive in computational terms, as potentially all the different design paths may be explored.

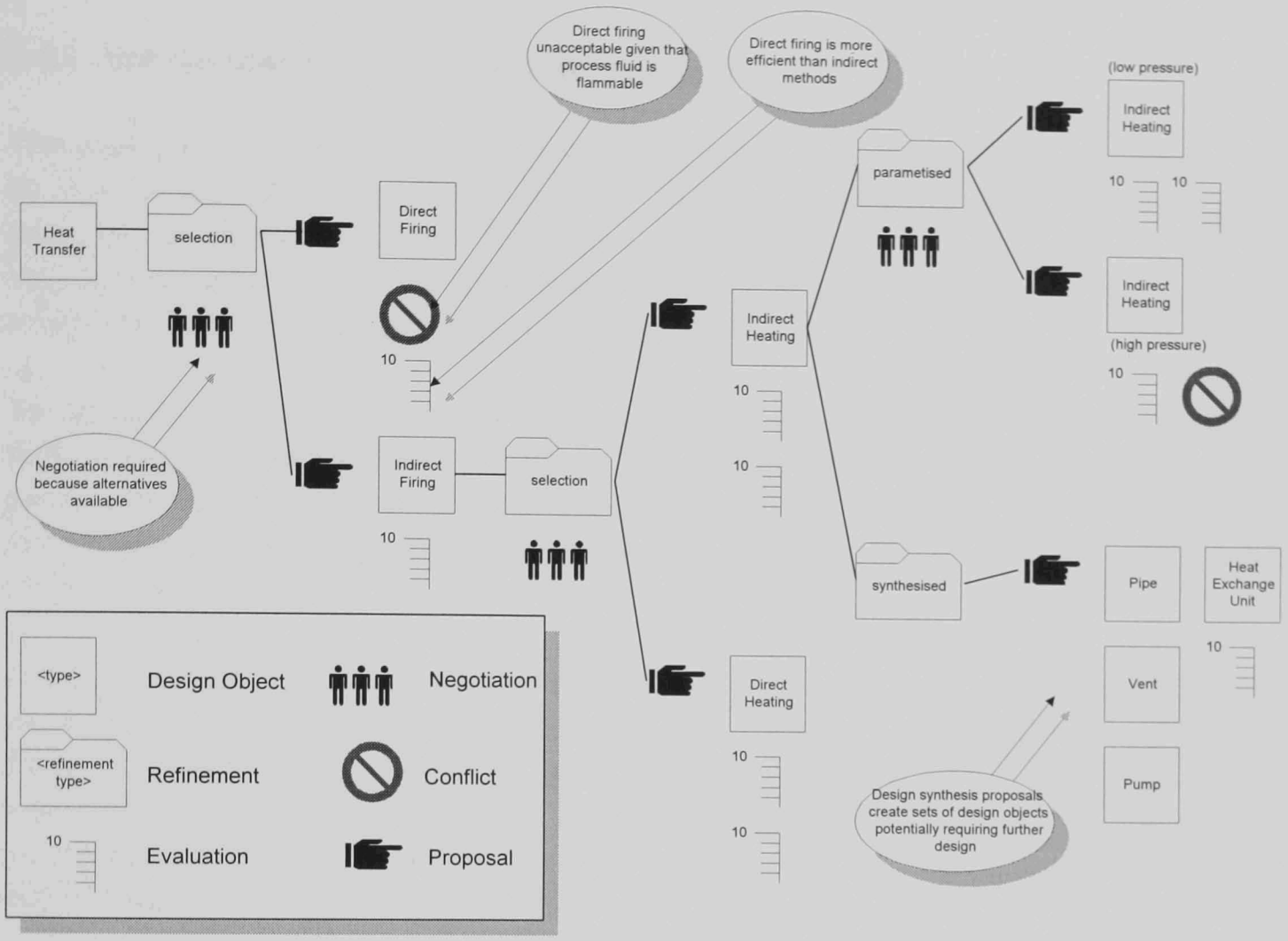


Figure 16. A simple depiction of the internal data model that will be formulated during the design process.

A solution to the problem of elaborating all the design paths is to account for the time spent in coming up with the design. In this case, if a lot of time is spent in coming up with a design, and this effort is accounted for in the evaluation of whether or not design should proceed along a certain path, the path on which most effort has been expended is likely to be chosen. One obviously does not want the design path to be explored further if the design quality has become really poor, in which case the solution may be unacceptable. Therefore an effective way is needed of specifying how much effort spent in design, affects the decision to choose the design path. This is what is termed, in CDEX, the *concurrent engineering* factor. If the factor is high (1), then time is not accounted for at all, and conversely if the factor is low (0), then time is very important and backtracking will only be performed in only the extreme cases where design is failing. This is analogous to the problem in traditional engineering design and is the problem which concurrent engineering is attempting to resolve. If the lifecycle issues and problems are not accounted for early in the design (backtracking required), compromises are made in order to save time and cost (i.e. the engineers will not take a step back in the design process if avoidable). In CDEX this would be modelled by a low 'concurrent engineering' factor. If on the other hand,

solutions are always developed with as much information accounted for as possible - which includes exploring several design alternatives if necessary - then backtracking will be a normal event and the 'concurrent engineering' factor will be high. This factor remains constant for any particular design case and is set by the user.

6.4.5 Internal view of design objects for example run

The example in Figure 16. is a very simple depiction of the internal data model that will be formulated during the design process. Initially the Heat Transfer design object is presented to the design space. This object may have been put forward for design by an engineering user, or as part of a larger design where the Heat Transfer object was synthesised as part of a process (e.g. from a Process Flow Diagram).

Those agents that have represented their interests in selecting a particular heat transfer system have put forward two proposals: Direct Heating, and Indirect Heating. As the proposals are presented as a selection refinement in the design space, agents that have noted their interests in evaluating the selection of a heat transfer system are requested to evaluate the proposals.

From the diagram you can see that a conflict has been recorded for the Direct Firing mechanism. The conflict in this case was a hard constraint, the agent found that a direct firing mechanism was inappropriate given that the process material that required heating was flammable. There is also an evaluation attached to the proposal. The evaluation indicates that the direct firing mechanism is suitable from an efficiency point of view (as direct firing in more efficient compared to the indirect method).

In this case the negotiation mechanism has not explored methods of 'fixing' the problem in direct firing. The negotiation mechanism makes this choice from a review of agent Evaluations, type of conflict, whether a specific conflict resolution strategy exists for avoiding the conflict, effort required in exploring for other solutions, and a number of other factors (see section 5.5).

Two proposals were presented for the type of Indirect Firing mechanism to adopt: Indirect Heating, and Direct Heating. The Indirect heating mechanism has proposals for both a Parametised Refinement and a Synthesised Refinement. In the CDEX system, Parametised refinements (i.e. sizing) have priority over other refinements, as the values of an object (e.g. maximum pressure, operating temperature etc) normally have an effect on the selection or synthesis design process. This is only to prevent abortive work, where another design process may make assumptions around missing design attributes. In the example shown, two proposals were put forward for the parametric design of the indirect heating mechanism. These two proposals accounted for both a high pressure, and low pressure solution to the heating requirement. The conflict shown for the high pressure solution was generated by a piping agent, who decided that due to the additional costs of special piping required to withstand the high pressure, the alternative would be unacceptable.

6.4.6 Combining agent assessments

The structured framework in CDEX enables knowledge to be applied to a design object that was defined as relevant to the objects parent. For example, knowledge concerning the PUMP object is applied when reviewing a centrifugal pump object. This mechanism allows for the application of generic knowledge in the framework. It also gives us the problem that an agent may provide more than one evaluation for a proposal, one from the perspective of the detailed proposal itself, and potentially one from a more general viewpoint if the object is part of a hierarchy, and the agent has more general knowledge at a level higher in this hierarchy than the object under review.

In order to resolve conflict, one must have a single viewpoint from the agent. In combining the viewpoints from the same agent when considering the above problem, the following issue is considered:

> - CDEX is based on the premise of a continual design refinement. The assessments (evaluations & conflicts) made throughout the design cycle are based on how effective the design solution is from the end perspective. Although the early assessments are not greatly reliable (denoted by an agent's confidence in a proposal), they are an assessment of how the agent thinks design will prove effective when complete. This enables us to access if design is progressing.
>
> Given this premise of continual design refinement, assessments made at a later stage in design by an agent regarding an objective, are going to be more reliable than the assessments made with knowledge of the more general items. More knowledge is available regarding an item (e.g, centrifugal pump) than what it can inherit from its parent (e.g. pump). It has therefore been assumed that an agents utility for an objective is a more 'informed' assessment, than an assessment made by the same agent for an object that was of a more general type.

Due to the fact that a proposal can be evaluated from a more generic viewpoint using some weighting function f() (e.g. for a pump), the same function will apply to a more specific design proposal (eg, a centrifugal pump). If a more specific evaluation function z() exists to evaluate a more specific design proposal (e.g. centrifugal pump) this would be due to the more information being available in the specific case. The assessment is more reliable, and therefore an overriding assessment. If the weighting function does not utilise any additional knowledge then the evaluation function is associated with the wrong design object and should be assigned to review a more general design case.

The combination of utilities for a particular agent are combined in the CDEX framework by the following approach:

- the objective utility in the combined result utility is the utility associated with the objective for the most specific case of the instance - i.e. the more informed assessment will be the correct one.

- Utility vectors not considered in any utility vector are denoted by 'nil'.
- The knowledge defined as the 'rationale' of all the proposals considered will be combined and presented as a single statement.

6.4.7 Functions used to assess negotiation parameters

The following functions are descriptions of functions in the CDEX framework that are used to determine the 'conflict classification' attributes denoted in Figure 14. These attributes are derived from an assessment of the proposals where conflict has been identified, and used in the selection of an appropriate strategy to resolve the conflict.

Anticipated conflict

definition: Anticipated conflict is the degree to which one believes there is likely to be conflict in the future, rather than the degree of the problem now.

Anticipated conflict is determined from analysis of the extent of conflict, and the belief in the conflict.

example use: If there is a belief that there will be a problem if a particular design route is chosen (irrespective of whether all the knowledge has been gathered), then a smoothing strategy (where more information is gathered) may be appropriate to access the extent of the problem further. A smoothing strategy is therefore appropriate where one is unsure (low confidence) of a conflict (low preference).

If the degree of belief in a conflict is high, then smoothing is not appropriate as more information is less likely to find a situation where the conflict is not apparent.

method: low belief + low preference = high anticipated conflict

modelled by: (1-belief) * (1-preference) = degree of anticipated conflict

Close to resolution

definition: How close the parties are to resolving the conflict for a specified proposal. They either all agree that the proposal is the best possible from all viewpoints involved, or the sacrifices made by each of the agents is reasonable (no single agent has lost to the others' benefit).

example use: If the parties involved are very close to resolving the conflict, then a compromise solution may be appropriate as neither party has much to lose

over the other.

method: 'Close to resolution' is 0 if there is a hard conflict associated with the proposal, i.e. someone determines the approach is not possible, irrespective of the sacrifice made by a party.

If all parties have achieved their maximum evaluation potential (preference / maximum-preference) then they cannot be closer to resolving the conflict.

If one party considers the proposal to be poor, and the others' consider the proposal to be good, then they are not close to resolving the conflict, irrespective of the number of agents who consider the proposal good.

If all agents consider the proposal to be poor, then they can be considered as being close to resolving conflict as the sacrifices made by each agent is large.

If there is only one viewpoint for a proposal, then 'close to resolution' must be 1 as there is no conflict.

modelled by: - Determine the list of revised preferences (the preference as a percentage of the maximum preference, or how much the agent believes in the proposal accounting only for the goals he considered).
- Determine the minimum and maximum values from this list of revised preferences. Note that a mean would not be appropriate as one does not regard the number of agents commenting on the proposal to be an indicator of how close the conflict is to resolution - the dissenting agent may just be more of a 'specialist' in a particular design area.
- calculate: close-to-resolution = min revised preference / max revised preference. One would expect that the degree to which how close the agents are to resolving conflict would not be greatly affected by small changes in opinion if the degree of difference is very large, therefore division provides us with a better model for having a poorer factor for 'close to resolution' with just small differences in the opinions.

Flexibility

definition: Determines the flexibility of a proposal. Flexibility is determined by the absence of hard conflicts, and if hard conflicts do exist, the degree of prominence specified by the agent who put forward the proposal. The degree of prominence is the difference between the agents belief in the proposal put forward, and the agents next best proposal if he has one.

example use: If an agent is inflexible on the proposal - i.e. does not want to budge on

their stated issues, then it is more appropriate to search for other solutions as a compromise does not seem likely.

The degree of prominence is an important factor for the agent to record if available. Obviously the higher degree of prominence identified (the difference between the proposed solution and the next best), the less likely the agent is going to like providing another alternative as the losses will be large. If on the other hand the degree of prominence is zero (the agent does not see the difference either way with regards to his two best proposals), he will not mind putting the other one forward if required.

method: We will assume an agent is flexible, unless there is a hard conflict. If there is a hard conflict, then its prominence will be reviewed. If the prominence is high (i.e. the next alternative available is not that good) then this reflects poorly on the flexibility. If the prominence is not specified and a hard conflict exists then assume that the agent is not flexible.

Note that prominence is only obtained from the main proposal (not evaluations and conflicts) as it is the difference that agent has between the one selected and the next best he has available to put forward. If other agents have proposals to put forward with different preferences then these will be put forward as separate proposals for review and considered separately.

modelled by:

```
if no hard conflicts exist then
 total flexibility
else
 if prominence specified then
        flexibility = (1 - prominence)
 else
        flexibility = 0
 endif
endif
```

Goal difference

definition: An indication of how closely the agents agree on the values applied in the proposal (not the preferences).

example use: If the agents conflict on the goals of discussion (large differences in maximum preference due to lack of knowledge regarding the object) then

further design can be performed to add detail to improve an agents viewpoint.

method: The bounds of the maximum preference associated with each of the evaluations/conflicts are determined and the difference is equal to the goal difference.

If an agent did not apply much value knowledge (low maximum-preference) in the analysis, and another agent applied all his values (indicated by a high maximum-preference) then the goal difference is high.

modelled by: Determine maximum and minimum values of maximum-preference for each proposal evaluation, conflict and the proposal itself. The difference is the 'goal-difference'.

Importance

definition: It is important to find a solution to a conflict with a proposal which most people like or a lot of time has been spent in its construction. These factors are accounted for elsewhere (time and preference) and therefore are not considered in this factor. Every proposal therefore cannot be considered to be important to resolve unless the prominence is low (i.e. its important to get right unless another solution is available to the agent which is almost as good).

example use: It's ok to compromise when reaching agreement is not important, i.e. in order to progress the design, if the importance of reaching a solution is low, any losses made when reaching a solution is not that important.

method: Everything should be considered equally important except the case where a prominence is specified. If the prominence is specified and the proposal is a lot better than the next best (a high prominence) then it is important to reach agreement. If another solution is available which is nearly as good (low prominence) then reaching a solution is not that important.

modelled by:

```
if prominence specified then
 importance =prominence
else
 importance = 0.5
endif
```

Issues involved

definition: Each agent shares the same structured utility vector (all second level objectives) albeit the low level objectives are likely to be different. 'Issues-involved' determines from analysis of the preference vectors, which values (objectives) were in the assessment of the proposal from all viewpoints.

example use: If only one objective was involved in the assessment and a conflict exists then a compromise is appropriate. When more objectives are involved then a strategy to drop the least important goals from the assessment becomes possible.

method: The preference vectors in the proposal, its conflicts and evaluations are all analysed to determine for each objective, whether it has been considered by any agent reviewing/presenting the proposal. A utility vector is returned, showing 1 for an objective if it was considered, and a 0 otherwise.

General preference

definition: Determines the average preference of all the evaluations, conflicts and the proposal itself for a particular proposal. It is to provide us with a general idea on how much all the parties involved approve in the proposal.

This function does not directly determine one of the identified conflict classification attributes previously described, although plays an important part in the assessment of the strategy and is referenced in the following section detailing the implementation of the conflict resolution strategies.

General confidence

definition: Determines the average confidence of all the evaluations, conflicts and the proposal itself for a particular proposal. It is to provide us with a general idea on how much confidence all parties have in the proposal.

This function does not directly determine one of the identified conflict classification attributes previously described, although plays an important part in the assessment of the strategy and is referenced in the following section detailing the implementation of the conflict resolution strategies.

6.4.8 The implemented conflict resolution techniques

The following describes the implementation of each of the conflict resolution strategies in CDEX. The definitions of each of the strategies have been covered in section 5.4. The models of each of the strategies has been derived from knowledge of the general rules in

which the strategy is applied. These rules have been taken and a general numerical technique applied that models the behaviour of the rule but with the benefit of a numerical weighting. These numerical weightings provide us with a fuzzy variable which indicate the degree to which the rule is met. For example, the Integrative conflict resolution strategy is more effective where the conflict is not close to being resolved. therefore close-to-resolution = 1 - close-to-resolution(proposal) (see function close-to-resolution above). The closer the agents are to resolving conflict, the less importance is ascribed to the Integrative strategy.

As mentioned previously, the selection of the conflict resolution strategy itself is a knowledge based problem. Each strategy determines a set of factors: potential to succeed, preference, confidence, and assessment of time. These factors are combined to give us a total weighting for the strategy in assessment of which strategy is best to resolve the conflict.

The following text covers the detailed design of the conflict resolution strategies as implemented in CDEX.

Consensus

Evaluate

- Determine how close parties are to resolving conflict. Abort if not close at all (i.e. hard conflict with confidence of 1).
- determine how flexible the agents are
- determine how important resolution is. This value is inverted as a less important problem favours consensus.
- potential-to-succeed is determined by:

```
(sum of (close to resolution * .3)
        (flexible * .2)
        (importance * .5)
)
```

- preference (revised preference from proposal)
- confidence (confidence from proposal)
- time (.02) if no hard conflicts exist.

If hard conflicts exist, then time is modified to account for problem in finding solution.

Apply — If the evaluation is best for a particular proposal, the proposal is chosen without any further analysis.

Compromise

Evaluate

- determine how close the agents are to resolving the conflict (close-to-resolution

(proposal))
- ensure that all agents involved in reviewing the proposal can exchange their values (utility vectors)
- check that the agent who supplied the proposal can put forward other alternatives
- If the agents can exchange goals, an alternative can be supplied, and there are no hard conflicts with a confidence of 1, then:
- determine the following factors
issue-factor = 1 if a single issue, 0 otherwise
(compromise better for single issue conflicts)
flexibility = flexibility (proposal)
importance = 1 - importance (proposal)
potential to succeed =
(close-to-resolution * .3) +
(issue-factor * .1) +
(flexibility * .3) +
(importance * .3)
preference = general preference of proposal
confidence = general confidence in proposal
time = 0.02 if no hard conflicts exist. Otherwise time is modified to account for potential problems.

Apply
- An average is taken of each utility for all viewpoints on the proposal (including conflicts), this becomes the 'compromised utility vector'
- The agent who presented the proposal is requested to supply another alternative, accounting for the 'compromised utility vector'.
- When the agent responds with a new proposal, other agents are requested to evaluate the proposal using the 'compromised utility vector'.

Generate alternatives

Evaluate
- If the agent who presented the proposal cannot generate alternatives, then this strategy will not work.
- determine the following:
close-to-resolution = 1 - close-to-resolution (proposal)
(alternative generation ok when not close to resolving conflict)
flexibility = 1 - flexibility (proposal)
(alternative generation ok when agents not flexible on the issues)
potential-to-succeed =
(close-to-resolution * .7) +
(flexibility * .3)
preference = general preference of proposal
confidence = general confidence in proposal

time = 0.05

Apply

The agent who supplied the proposal is requested to supply another.

Abandon less important goals

Evaluate

- determine the following:
 issues-involved = issues-involved (proposal)
 issue-factor = 1 if more than one issue involved, 0 otherwise
 flexibility = flexible (proposal)
- if the agent cannot supply other alternatives, a hard conflict exists with a confidence of 1, or there is not more than one issue involved then the strategy is not appropriate.

 potential-to-succeed =
 (issue-factor * 0.5) +
 (flexible * 0.5)
 preference = general preference of proposal
 confidence = general confidence in proposal
 time = 0.02 if no hard conflicts. Time is modified to account for potential failure regarding hard conflicts later in the design.

Apply

- For the proposal, conflicts and evaluations, the least important objective (lowest utility) is determined, the value for the objective is set to zero (drops the goal from consideration), and then adds this loss uniformly to the other utilities to maintain the differential (i.e. sum of 1). These new utility vectors are recorded for each agent.
- The agent presenting the proposal is requested to supply a new proposal using his new modified utility vector.
- When the new proposal is received, each agent involved in the initial proposal evaluation is requested to supply an alternative using the new modified utility vector determined earlier for that particular agent.

NOTE: Only the agents that presented a viewpoint in the initial proposal will have a chance to put forward their viewpoint on the new proposal.

Integrative

Evaluate

- determine the number of issues involved (issues-involved)
 issue-factor = 1 if more than one issue involved, 0 otherwise
 flexibility = flexibility (proposal)

close-to-resolution = 1 - close-to-resolution (proposal)
- If the agent who presented the proposal cannot provide alternatives, or the number of issues not greater than one, or a hard conflict exists with confidence of 1, then an integrative strategy is not appropriate.
potential-to-succeed =
(flexible * .2) +
(close-to-resolution * .8)
preference = general preference of proposal
confidence = general confidence of proposal
time = .02 if no hard conflicts exists, otherwise time is modified to account for the potential for design failure in the later design stages.

Apply

- The new utility vectors for the proposal, evaluations and conflicts are determined. This is done by

(for all agents)
- determine lowest-utility in agents utility vector
- an issue is important if it is greater than the lowest- utility + (lowest-utility * utility-prominence)
- an important-utilities vector is constructed with a 1 indicating an important utility, 0 otherwise.
utility-prominence = 0.5 in these cases but can be modified if necessary.

(for all important-utilities)
- construct a combined-important-utilities vector with a 1 indicating it is important to at least one of the agents, 0 otherwise.

(for all agents)
- For each utility in the agents utility vector, if it is not globally important (determined from combined-important-utilities), then reduce the agents utility by (utility value * reduction-factor).
- reduction-factor is 0.5 in these cases but can be modified if necessary.
- The total reduction in the agents utility is recorded, and then this total reduction in value is assigned to those utilities that are important from the global perspective (combined-important-utilities). The value of the total reduction is spread equally among the utilities important from the global perspective.

In more simple terms, one can determine those objectives which are important to everyone, and increase the importance of those by reducing those objectives that are not important to everyone (i.e. account for the global perspective).

- A new proposal is then requested from the agent who initially presented the proposal, using the new modified utility vector determined above.

- When the new proposal is received, the agents that initially presented a viewpoint in the initial proposal, are then requested to supply an evaluation of the new proposal, using the agents modified utility vector determined previously.

Smoothing

Evaluate

anticipation = anticipated-conflict (proposal)
difference = goal-difference (proposal)
- if the anticipation factor is not greater than the minimum value acceptable, in these cases 0.5, then do not consider the strategy to be appropriate. The minimum level of anticipation can be modified. Smoothing is not appropriate where a hard conflict exists with a confidence of 1.
potential-to-succeed =
(anticipation * .7) +
(difference * .3)
preference = general preference from proposal
confidence = general confidence from proposal
time = 0.05 (low iterative) if no hard conflicts exist, otherwise time is modified to account for the potential of design failure in later design stages.

Apply

- The proposal is requested to progress its design. Normally this does not occur unless the proposal is accepted. The negotiation mechanism is informed that the strategy wishes to review process. This will prevent the design from continuing without permission (see detailed design permission-to-proceed) from the smoothing strategy. i.e. only if it considers things to be improving will it allow design to continue.
- When further design is requested, check that the confidence in the solution has improved, while accounting for the quality of the design. If anticipation of conflict is low then the knowledge of the problem has increased. Therefore the previous estimate of preference is compared to the preference that is actually being attained by the more detailed design. If design is improving, then design is allowed to continue, otherwise further detailed design is prevented, and the current position is reviewed (re-negotiated).

Majority rule

Evaluate

- determine if the majority support the proposal, regardless of any conflicts (with the exception of hard conflicts). Majority support is determined by more agents reviewing the preference of the proposal with a belief greater than .5 than those that do not.
- determine the following:

close-to-resolution = 1 - close-to-resolution (proposal)
(majority rule good for when parties are not close to resolving conflict)
flexible = flexibility (proposal)
importance = (1 - (4 * (x * (1 - x)))) where x = importance (proposal)
(majority support is effective where the problem is either not very important - it does not matter which one is chosen, or where it is very important to resolve - a decision has to be made)
potential-to-succeed =
(close-to-resolution * .3) +
(flexible * .3) +
(importance * .4)
preference = general preference of proposal
confidence = general confidence in proposal
time = 0.02 (single shot) if no hard conflicts exist. If hard conflicts exist, then this time is modified to account for potential of design failure in the later design stages.

Apply

The proposal reviewed is chosen for further design. No further analysis/review is performed.

Specific CR strategy

Evaluate

- analyse each of the conflicts in the proposal, and determine if any agents have suggested that they can solve them.
- If a conflict with a proposal that an agent can resolve:
potential-to-succeed = 1
(i.e. if an agent is aware of the problem, the resolution is likely to be a good solution)
preference = general preference for the proposal
confidence = general confidence in the proposal
time = 0.02 (single shot)

Apply

- Inform the agent to present a new proposal accounting for the conflicts which the agent identified as those he could resolve.
- When the new proposal is received, request agents to evaluate the proposal as normal. Any agent can evaluate the proposal in this case.

6.4.9 Avoiding repetitive application of resolution strategy

A strategy is not applied to the same proposal again, although it may of course be applied to the proposal generated as a result of the previous application of the strategy. For

example proposal A is put forward, a conflict is identified, the compromise strategy is selected to resolve conflict, and proposal B is put forward as a compromise on A. When the negotiation mechanism re-accesses the list of potential proposals to choose from, it may again find that proposal A is generally more preferred than the other proposals, but that the agents do not agree on it being the best proposal. It would be inappropriate for the negotiation mechanism to compromise on proposal A again as one would get the same result as before - another proposal equal to proposal B. The negotiation mechanism therefore does not allow the same strategy to be applied to the same proposal a second time. The mechanism does however allow the same strategy to be applied to a proposal derived as a result of the applying the strategy previously. In this case, one could apply the compromise strategy to proposal B, and get the result proposal C which is a further compromise on proposal B, which is a compromise on A. If the compromise strategy is effective one should be moving closer to resolution, although it must be noted that at each stage conflict exists the strategy is still competing with other strategies that are putting their case forward for resolving the conflict.

Potentially there could be problems with the strategy outlined above if the degree of compromise at each stage is so small. If this is the case, the strategy is likely to be continually applied to each of the proposals derived from applying the strategy and therefore there is considerable iterative application of the same strategy with little result. To avoid this problem, as a strategy is applied to resolve the problem, the time in which the strategy is expected to resolve conflict is doubled. This is implying that given the strategy has failed previously, if the strategy is again selected to resolve the conflict it may fail again and therefore take twice the time as it expected previously to resolve the conflict. Each time the strategy fails, the time is increased, and the proposal becomes less likely to accepted by the negotiation mechanism because of the time it expects the strategy to resolve the conflict.

6.4.10 Ensuring design proceeds as expected

Permission to proceed is requested if the agents reviewing the more detailed design considers that design is not proceeding as well as expected. 'As well as expected' is known by the preference that was associated with the proposal earlier in the design phase that lead to the particular detailed design route being accepted. Obviously at the time the design route was selected, that design route looked the most promising and therefore as long as the preference for the solution remains as high as that proposed, the design route is acceptable. When the quality of design falls below that expected earlier in the design phase, then obviously the decision made earlier in the design to take the particular design route in question has to be reviewed. It may be the case that the design route is still the most appropriate route to take, or it may be that another route now looks more promising given the fact the current design path taken is not as promising as first thought. 'Permission to proceed' is a request from a more detailed design route to continue design if it is found that the route selected was not as promising as first thought. This is a simple

search approach utilised in many fields of artificial intelligence and is appropriate in this case as long as agents are not too optimistic. If agents are too optimistic about their early design assessments, then more detailed design scenario's are not likely to produce results as fruitful as first thought, and CDEX will be continually finding itself refusing design to continue down a path and trying other alternatives. This would be similar in nature to breadth first search - all the opportunities will be explored - and the computational overhead in such a strategy would be very high. If on the other hand, agents are pessimistic, it may lead to fruitful design paths not being explored and therefore the design quality may not be as high as it could be. The approach in CDEX is for the agents to aim to be as honest as possible in assessments of a design path which in some respects is helped by the more prescriptive approach to identification of ideals provided by utilising the objective hierarchy and utility theory.

An issue addressed in this research was combining assessments for a design process that has been synthesised. In the selection and parametric design process, any request to proceed from a more detailed design process will be a complete more detailed assessment of the whole solution. For example, if an atmospheric tank has been reviewed, and permission to proceed design is requested as a result of analysing the atmospheric tank. then if the atmospheric tank was selected as a storage mechanism, one can be sure that the assessment of the atmospheric tank is a more refined assessment than that made for the storage mechanism as more information is available (as well as the knowledge concerning a storage tank would have been again applied to the atmospheric tank). For a design that has been synthesised however, a 'permission to proceed' request has an assessment for only a part of the design, and not the complete picture. It therefore does not seem logical to review the request to continue a small part of the design without an assessment of the complete design available. However, if it were to be considered how one would deal with a number of different assessments of a synthesised design, a logical approach would appear to weight the design by the worst review as:

> ...an operating plant is only as good as its worst performing component
> ...a design is only as safe as it's least safest part
> ...if only a small part of the plant needs constant attention from the operator, the plant as a whole will always need an operator
> ...etc.

CDEX therefore evaluates each design assessment individually, and always requests permission to proceed if design is not progressing as expected. The exception to this rule is where time is important. CDEX accounts for the design effort spent on a particular design option and accounts for this in the assessment of whether design should continue. If time is an issue - the engineer requires a quick solution and not necessarily the best - then more emphasis (a greater weighting) is placed on accepting a design path even if the solution is not turning out as expected. Time also plays an important role in selecting a conflict resolution strategy that is most suitable if time is short.

6.5. Requirements of a tool to cooperate in the framework

The agents in the framework present a design viewpoint of the engineers involved in the design case and the agents were developed to essentially record the knowledge identified in the design case. They could be considered knowledge based, although this is dubious due to how the results are derived. The results of each design requirement or evaluation were fixed as the response provided by the engineer in the design case was known . The result itself was not deduced from evaluation of a number of rules, as it was not known how the engineer came to his conclusions - it was just known what the conclusions were. This research concentrated on proving that the conflict between the different viewpoints could be resolved and a rational design approach adopted. This required the knowledge of the formulated opinions of each agent - not a knowledge of how the opinion was formulated. However, it was identified in the requirements for the framework that disparate technologies should be able to cooperate. This section details how disparate technologies could work in the proposed framework.

6.5.1 Provision of external services through a 'Wrapper'

In order for a software system to cooperate in the framework, a 'Wrapper' is required (see section 4.6.3 The Engineer). A Wrapper translates the inputs and outputs of a software system so that it can communicate with other software systems. Several varieties of Wrapper can be implemented in CDEX from very simple mechanisms to basically pass information from the framework to the software, or to very complex mechanisms that control the external software systems and formulate responses to the results produced by the software.

6.5.2 Basis requirements of external software systems

In CDEX the external software systems, in order to cooperate in the framework have to conform to a specific set of requirements. These requirements are identified below. Issues pertinent to the Wrapper providing the necessary functionality is also described.

i. present solutions and evaluations as proposals

   The wrapper can take the output from external packages and place the output in a proposal for evaluation by the framework.

ii. the proposals presented should have identified preference vectors with an associated total preference for the solution from the agents perspective

   This is one of the more restricting requirements, along with the requirement for a confidence factor as described below. It may be considered that this implicitly effects the design strategy adopted by the external software system, but this not necessarily so. The wrapper could provide significant functionality. For example,

the wrapper could take the proposed solution by the external software system and either:

a/ request the users to identify the preference and confidence in the solution

b/ evaluate the solution developed by the external system with the use of local evaluation functions to the wrapper, and local preference criteria.

c/ use constants for the preferences

Solution b/ above is, considered in most cases, a reasonable approach although the evaluation criteria is outside the scope of the main application which is where such knowledge would be useful. Solution a/ is not really acceptable as it is not reasonable to expect immediate responses from an engineer for quick what-if test approaches to design approaches by engineers early in the design phase. If time is not critical, and the design paths are ones which the engineer considers to be the path accepted then requesting different engineering disciplines to evaluate the design may not be a problem. The use of constants (solution c/) could be applicable where there are a fixed set of choices and the engineer has specified the preferences for each choice beforehand. The wrapper could provide a lookup mechanism whereby when the external software system decides on a solution, the appropriate pre-defined constants for the solution could be looked up and provided along with the proposal.

iii. each proposal must have a confidence associated with the agents belief in the design path turning out how it expects

(The same issues as with preferences - see above ii.)

iv. present evaluations that account for the quality of the design path (end result) from its perspective

Most of the issues are related to those with determining the preference and confidence of a proposal (see above). Additional issues are with regard to how a design is assessed from the viewpoint of the complete design path, as opposed to a result of reviewing just a couple of design issues. The wrapper in this case could keep a track of how the external system reviews each developed solution, and where only a small part of the solution path is reviewed, the wrapper could combine the view with the assessments previously made for the design from that software agent, thereby providing a complete view of the design path that could be obtained from the associated software system. This approach would not be needed however if the wrapper could evaluate the whole solution.

v. present proposals as objects in the style that has been agreed (i.e. centrally defined knowledge representation)

The wrapper will act as a translation mechanism to take the design produced by the external software system and translate it into the necessary design objects and

presented as a proposal.

vi. be able to interpret the objects defined in the centrally defined knowledge representation format.

(see v. above)

vii. be able to express parts of the design that the agent is interest in (evaluating & designing different design components).

This would be done by the wrapper. It is expected that the capabilities of an external software system do not change throughout the design process and therefore these 'interests' are defined only once, when the wrapper is developed.

viii. be able to identify when a solution cannot be provided by the agent, or that it cannot evaluate a solution in which it expressed an interest in evaluating.

The wrapper would have to be able to determine when the external software system can not provide a response to a request and inform the framework when appropriate.

ix. work off-line

In a 'what-if' mode it is appropriate that repones to a design requirement are provided to an engineer reviewing an alternative quickly. It is also important that other engineers are not bothered by many different requests to review solution paths in the what-if design mode, as much time could be spent reviewing unfruitful design paths. Many systems do not provide the ability for off-line working and often provide access to functionality through its interface. This however is changing though the use of API's (application program interfaces) and information exchange standards that enable external programs to interface to the softwares functionality. Notable developments in this area have been Microsofts OLE[3] and DDE[4].

A technology to support this area where external software systems cannot provide off-line response is case analysis. Case analysis would enable design proposals to be presented from knowledge of proposals put forward in the past. This however requires a set of case histories and the technique presents its own problems. The responses can also be provided by an expert wrapper that can present proposals for

---

[3] Object linking and embedding. A technology to provide an interface (essentially a set of functions) to the softwares functionality without having to use its interface.

[4] Dynamic data exchange. The ability for a program to send requests (specific actions) to another application. Other applications can be requested to perform specific actions. The applications interface does not have to be visible, but the application must be loaded.

a requirement. This would not have to be vastly complicated and put forward the most obvious proposals from a quick analyses of the problem. The proposals put forward by the wrapper can be presented with the requirement to the appropriate discipline at a later date for review. In this way the engineer can review the performance of the wrapper in presenting default proposals, and correct behaviour if proposals put forward for a requirement are not appropriate.

6.5.3 Optional abilities of external software systems

The following additional requirements are optional, that is, the agent is not expected to conform or apply the following functionality:

i. be able to propose alternative solutions

   Alternative solutions are requested where the one previously put forward was not immediately accepted. The external software mechanism would either have to have the capability to provide another alternative, or the wrapper would maintain a list of options developed by the external software and provide these to the framework when requested.

ii. be able to assess a proposal from another agent's viewpoint (i.e. exchange preference utility vectors).

   If a solution was reviewed by the wrapper, the wrapper could take care of re-evaluating the proposal using the new evaluation criteria supplied by the framework. If the application performed the assessment, the application would have to have the necessary capability of allowing new criteria to be specified.

iii. be able to identify what is wrong with a proposal by using a keyword that is descriptive of the problem identified.

   It is expected that the keywords will be developed alongside the central object model that is the definition of design objects which the agents must conform to. These keywords need to be explicitly defined in order to enable interpretation by disparate software systems. It is imagined that these keywords will be similar in nature to the HAZOP guidewords (e.g. more flow, high level) which are explicit keywords applied to a detailed design in order to identify potential problems. These keywords are generic in the sense they are applicable to all types of design object (although in a few cases a keyword may not be applicable). An external system that can put forward alternative proposals based on knowledge of a problem is likely to be knowledge based of some sorts and therefore a translation from the keyword to a format required by the external software program is required.

iv. Able to identify a prominence of the proposed alternative over other alternatives.

The negotiation mechanism can account for a 'prominence' of a solution in selecting a strategy to resolve conflict. The 'prominence' is an indication of how much better the proposal put forward is an improvement on the next best available. Issues here are the same as i/ (provide alternative solutions) and ii/ (determine preference) documented above. The value can only be determined if the selection tool can provide a second alternative. This second alternative may have been provided with the first alternative in a list (in which case the wrapper remember the list of proposals and weightings and provide them when requested). If a complete list is not presented to the wrapper in the first case, the software would have to keep track of which results (proposals) had been put forward, and therefore not present them a second time.

v. To be able to identify objects that have been accepted and deleted.

The framework provides various information messages with reference to when a proposal has been either accepted or deleted. If this knowledge is important to the external system the wrapper would have to translate this event into a message to the external system.

6.5.4 Example programs which require the services of a 'Wrapper'

The following is an example set of applications, broken down by specific design classification, that would require wrappers in order to cooperate in the CDEX cooperative framework.

Tools for the parametric design function (sizing)

- spreadsheets
- specific (inflexible) code modules (programs) for design sizing
- mathematical systems with backup database capabilities. e.g. determining the material of an object.

Tools for design selection

- database systems with sizing and lookup capabilities. Many equipment manufacturers produce their catalogues on disk so that customers can lookup products electronically. Some provide selection capabilities, either on simple sizing rules to more complex knowledge based approaches.

Tools for the design synthesis

- database of pre-defined system layouts/components. In CAD engineers will sometimes cut and paste standard systems into other designs. Manufacturers now

exist for supplying complete systems, heat exchange systems for example. These systems have defined components and have the benefit of experience being tested in the field.

- CAD. Given a requirement for a system, an engineer will produce a system that 'realises' the requirement. This is design synthesis, and the engineer can play a part in the CDEX framework by presenting his design as a synthesised proposal for a more general requirement.

- Process Flow Sheets are often standardised (e.g. Nearly all ethylene plants in the world are based on the same flowsheet) and therefore designs can be stored, sized by the process flowsheeting programs, and presented as a design synthesis to a specific design requirement.

6.6. Summary

The chapter has provided the detail necessary to understand the CDEX framework from a technical perspective. The theoretical basis of the framework described was covered in chapter 4. This chapter has covered the important elements of the design in a way which aids understanding. A more detailed design which covers the object models, state transmission models (object behaviour), the object parameters, and detailed method descriptions can be found in Appendix K. The CDEX grammar that enables agents to cooperate in the framework was described. A more detailed description of the grammar can also be found in the appendix (appendix L).

The negotiation mechanism implemented within CDEX was described in detail, and is in essence the implementation of the theoretical foundations of negotiation described in the previous chapter. The mechanism reviews particular attributes that can be determined from analysis of the conflict context. These attributes are the 'framework attributes' and identify things like preference of choice (how much better the agent's proposal is better than his next best assessment) and the degree of conflict. From these attributes one can determine the 'conflict classification attributes'. These attributes directly impact on the choice of the resolution strategy that will be selected to resolve the conflict (e.g. how close the parties are to resolving conflict will impact on whether a compromise strategy is selected to resolve conflict). Many issues were discussed with regard to how the negotiation mechanism operates. These issues covered how assessments were used to select a strategy, how agents assessments were combined if multiple views were presented, how the mechanism avoids iterative application of the same strategy, and several other issues.

In order for agents to cooperate in the framework, the agents' must conform to a specific set of requirements. These requirements are intentionally designed to be non-restrictive

in order to allow disparate technologies to work together in resolving a design problem. A program known as a 'Wrapper' is described that will enable existing systems to cooperate within the framework. Obviously compromises have had to be made with regard to how the mechanisms must operate. These compromises and issues regarding how existing and new technologies are developed to enable cooperation is described.

The following chapter describes the specification and test of an example design scenario between two design engineers with different objectives. The design scenarios will aid in understanding the CDEX system in execution and how agents interact throughout the design.

Chapter 7. Design Scenario

7.1. The Approach

The objective that CDEX is attempting to prove is that disparate systems with different capabilities, design approaches and expertise can cooperate together on a single design and develop rational solutions. This is a difficult requirement to test for, as many subjective weightings are being considered and the correct 'rational' solution in most cases is difficult to determine.

The research carried out a design study with two engineers designing a small part of a chemical plant for the storage and transfer of toluene to a distillation column. The two engineers each had an appreciation of the other's job and there was considerable overlap in skills. The names of the engineers have been changed in the discussion to maintain anonymity. The design discussion was recorded and transcribed and is documented in this chapter. An analysis was then performed on the information in order to get the information into a format that could be utilised in the CDEX framework. The design objects were identified and classified, the engineering rules for both engineers' documented, and the engineers' objectives organised into a format appropriate for analysis (the objective hierarchy). Two agents were then developed to encapsulate the knowledge of each engineer in the design case. The agents were then run in the cooperative CDEX framework and a result generated. Two complete scenarios were run through the framework, one run was to identify the best solution from the life-cycle perspective, the other to develop the quickest solution. These results were produced in the form of a textual design tree and are available in the appendix. Both sets of CDEX results were analysed to identify deviations and abnormalities in the approach taken in the design, as compared to the approach covered in the discussion. These issues are documented in this chapter.

It was not always clear in the design case what was the best design approach. In most cases not enough information was available to make a concrete decision on a particular approach. It was possible to gauge which solution was generally thought popular, as both engineers would spend more time evaluating the solution and questioning potential problems with the approach. It is assumed that the engineers would not waste time discussing solutions that they considered were not appropriate. The results were analysed with this problem in mind.

An important test was to ensure that the set of objective weightings were fixed and were the same for all decision points in the design. Only one set of objective values can be defined for each agent. Through finding a set of values that enable the correct action to be taken at each decision point, one can be sure that the utility values ascribed to the objectives are more in tune with the engineer's values, and also show that the framework - together with the negotiation mechanism and conflict resolution strategies - can be trusted

to make sensible design decisions, utilising the values and knowledge from disparate knowledge sources.

7.2. The Meeting

7.2.1 Content of the Design Meeting

The diagram below depicts the general picture of the plant developed during the design discussion. It is not an exact picture of what was finally decided in the design session as several of the issues involved in the design were discussed but a conclusion was not reached. Obviously difficulties exist in a 'one off' design meeting such as this as in a real life situation the engineers will have to go and search the literature, discuss the issues with other specialist engineers, and consult drawings of the site layout, local population density and many other sources.

The design problem was essentially to provide toluene as a feed product to a distillation column. From the diagram it can be seen that toluene is tankered in and off-loaded using a nitrogen pressure supply to the storage tank. There is a stream to the still as expected, and a re-cycle back from the distillation column for unused feed product. The re-cycle was specified as a design requirement. The storage tank has a vent, enabling the flammable vapour to be sent downstream to a thermal oxidiser for combustion. A bypass line is included in this configuration in the case where the thermal oxidiser is not working, thereby providing a backup form of combustion - albeit not as effective. Several options were discussed in the meeting on the design of the suction system to solve the expected back pressure problem due to the use of the thermal oxidiser. This suction system is not shown in the diagram.

The design requirement provided as a basis of the meeting was a simplified process flow diagram showing the basic process units. Although the requirement did show a storage mechanism, a re-cycle line from the distillation column, and various other items, it did not restrict the type of design that they could have developed and they covered many of the potential alternatives (e.g. they considered non-storage solutions such as tankering and a pipeline).

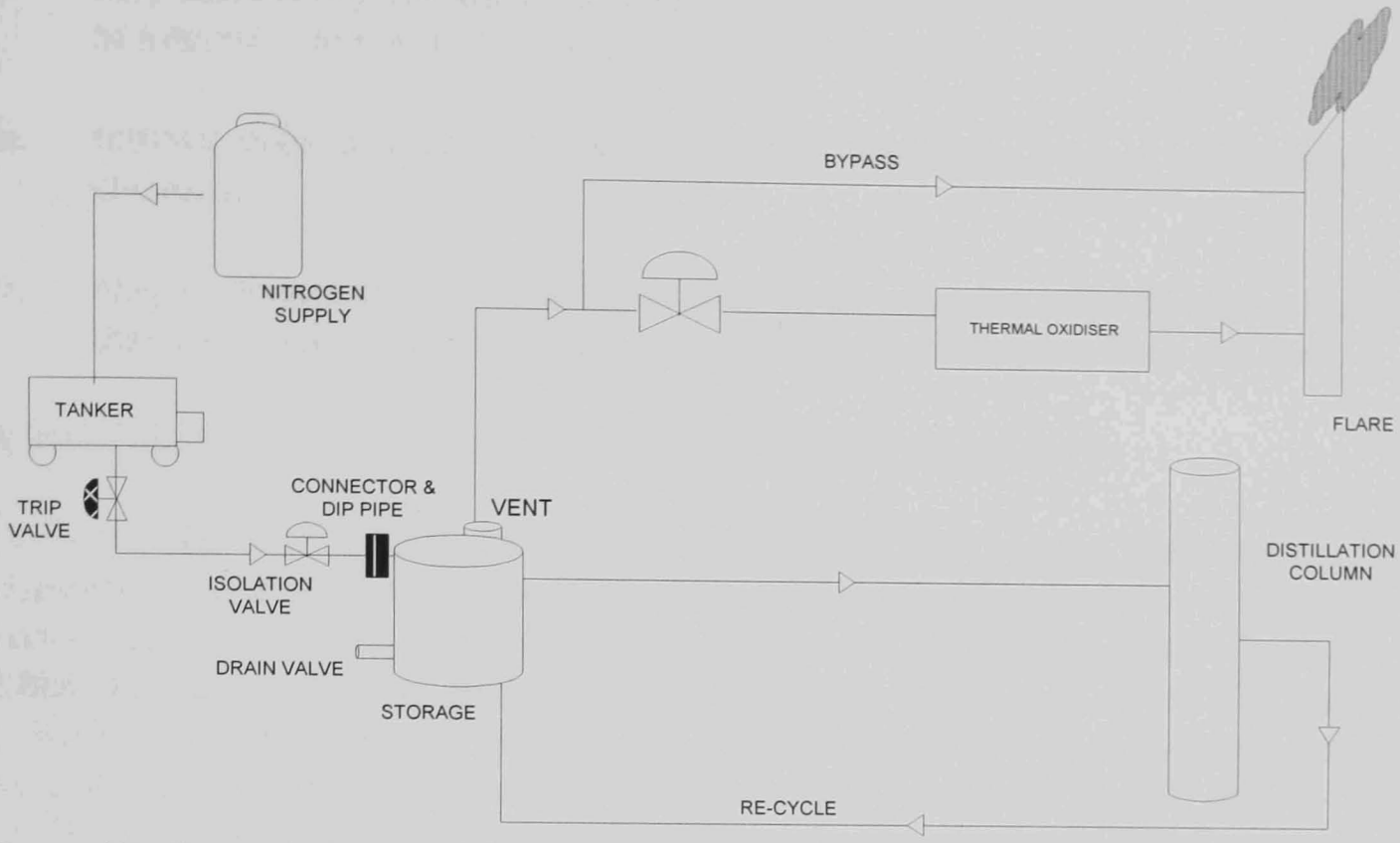


Figure 17 General picture of the system developed in the design scenario

The transcript of the vessel design meeting can be found in Appendix N. An understanding of the design case is important in understanding the results produced by CDEX. The design developed is similar in structure to the scenario above, although the above diagram does not show the various design avenues explored.

7.3. Content Analysis

7.3.1 Identifying objects of interest in the design case

In order for the two agents to communicate and share design information a common language is required. This language is formulated as a class diagram, using the standard object oriented development principles. The objects identified in these diagrams then provide the basis on which engineers provide design information and proposals (see chapter 6.2).

The identification of objects in the example design case adopted one of the techniques applied in object oriented analysis, which is to go through the text underlining all the objects. This list provides the basis for further OO analysis.

The next step is to review each object in the list and apply the following general criteria:

i. remove objects which are really attributes of objects rather than objects themselves. e.g. 'air' can be a value of 'fluid description' of object fluid.

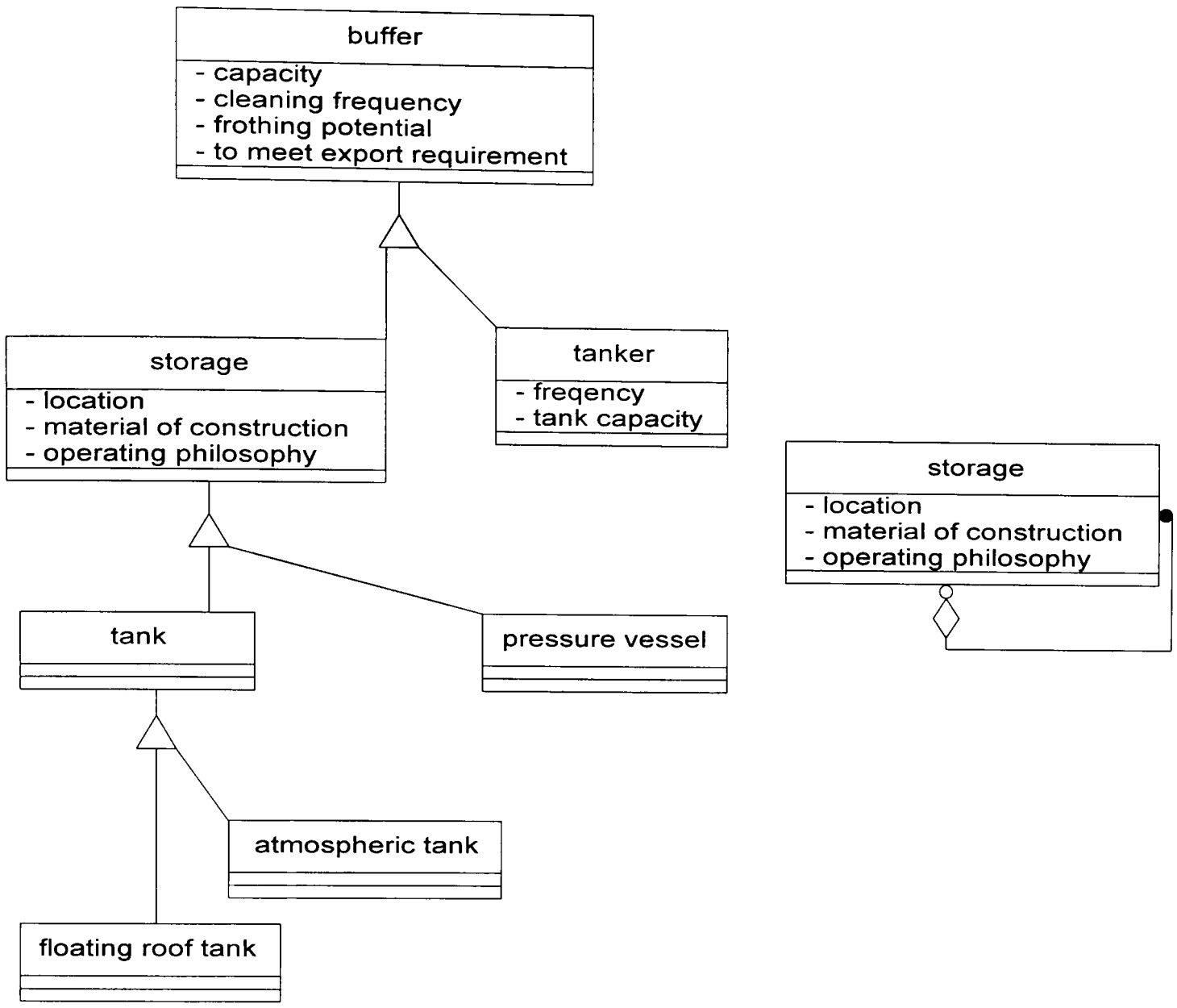
ii. only include objects which are required to model the plant design, and which are of interest. Do not include concepts outside the system.

iii. remove objects used under another term. Example, fluid is used instead of chemical.

iv. remove objects that are modelled in more direct ways (i.e. what the object is rather than its function). e.g. a recycle loop is a pipe from the plant back to a tank.

A list of objects identified in the design case is listed in Appendix D.

The next step is to identify the more general higher level classes that have not been identified in the discussion, but are inherent from the objects identified. For example, there exists a pressure vessel, a floating roof tank and an atmospheric tank. These are all different methods of storing product, therefore there is a STORAGE class. There are also different types of storage, for example the TANKER object (which delivers the product) but has different behaviour than what would be expected of a STORAGE object. These STORAGE and TANKER objects are therefore grouped under the BUFFER process (any item of equipment that holds fluid as its primary purpose). As mentioned previously, identification of this hierarchy is not a prescriptive task, and is in some cases more of an art than a science. As long as the agents both share the common representation and understanding there will not be a problem. If the hierarchy however is very flat (not many groupings or classifications) there will not be many general rules, but rules that are very specific to the design object in question. This will therefore require greater effort in analysis and ultimately more rules will be required to be defined (i.e. there will be many specific rules that could have been handled by a more general rule). For example, there may exist a rule for a storage tank: 'If the product is X, and X is a category D product, then we cannot store more than 2 tonnes on site'. If you consider for a moment that a STORAGE type does not exist, one would have to duplicate this rule for the PRESSURE VESSEL, the ATMOSPHERIC STORAGE TANK, and the FLOATING ROOF TANK, because in all cases the rule applies. Through having the classification of 'STORAGE', the rule can be associated to objects of this type (which encompass all the types of storage just mentioned) and therefore defined only once.



The attributes for the design objects can be determined by going through the design case and noting properties of the objects determined to be in the CDEX design environment. Judgement must be applied as to whether the attribute is local to the object to which it was initially associated or to one of its parents (higher classifications). These properties are also listed in Appendix D.

7.3.2 Determination of objectives in the design case

The objectives of each of the engineers in the design case must be understood in order to provide a means through which negotiation can take place. These objectives identify a preferred 'direction' that an engineer takes in his/her approach to design, for example, an engineer may veer more towards the safety than the cost angle. The objectives identified in the design case are listed in Appendix E. Note that there is some overlap with the Actions and Requirements list which is expected. Each objective is tagged with the engineer who showed the concern for a particular objective.

The objectives can then be classified, grouped and re-worded if necessary. For example,

when Roger notes that one should be able to cope with reverse flow from the still, and be able to cope with a loss of nitrogen pressure control to the storage tank, he is considering the objective of the process to cope with adverse conditions. An element of judgement is needed here to construct the hierarchy as there is no real formal approach. The next stage, and one that is more subjective, is to assign utility weightings to each of the objectives to account for how much a particular objective is important in achieving its parent objective.

The objectives are listed in Appendix E in the order that they appear. Some objectives are repeated, but it provides a guide to understanding the importance to which the engineers subscribe to particular objectives. For example, if Jeremy always critiques a proposal due to a safety issue, and Roger tends to go straight for the cost implications, then it could be assumed that Jeremy considers safety more important than cost, and Roger considers cost more highly than safety. This assumption may however be incorrect (and is possibly likely to be) as when the scenario is discussed Roger may be playing the 'devil's advocate' regarding a safe design knowing that Jeremy will highlight the important safety considerations. At a more personnel level, outside the bounds of a group design, Roger may tend to comment more on the safety angle than cost. The weightings utilised in the scenario are therefore subjective although the implications on the validity of results is not important as the weightings can be easily changed in order to enable us to progress down a design path which both engineers agreed on. The important test however is to provide a single set of weightings that are applicable for all decision points in the design that enable the agreed design routes to be traversed. Only one set of values can be defined for each agent. Through finding a set of values that enable the correct action to be taken at each decision point, one can be sure that the utility values ascribed to the objectives are more in tune with the engineer's values, and also show that the framework - together with the negotiation mechanism and conflict resolution strategies - can be trusted to make sensible design decisions, utilising the values and knowledge from disparate knowledge sources.

It is possible however that a different design route is taken even if one has accurate assessments of the engineer's values. Humans are not always consistent in their application of values, and when many issues are under consideration - in order to simplify the decision - certain values may be dropped. These differences and justifications for the potential discrepancy are documented in the test results.

The accuracy of the utility weights is not expected to be a major issue as long as the general qualitative factors are accounted for (i.e. we consider cost equally/less than/more than with safety). Using the pure logical terms TRUE and FALSE, one can easily state that safety comes above cost, but it is not possible to specify the degree to which this may be the case. There will be a point where the risk is minimal compared to the cost of an expected problem and the cost of reducing the risk further. The utility weights enable us to have a 'fuzzy' model of the values involved in the decision making process. Issues are only likely to be identified regarding the utility weights when there is a very difficult choice to make (i.e. both options are similar in preference). Small inaccuracies in the weightings with these difficult choices will likely lead to what can be considered a

'random' choice among the alternatives because the inaccuracies are not known, and secondly it is difficult to determine whether the approach taken was right or not. From analysis of the design case however, one does not appear to have these kinds of decision, as usually a choice is based on more fundamental problems with other available alternatives (e.g. cannot store toluene in an atmospheric tank). The objective hierarchy with associate utility weightings for the design case is noted along with the objectives in appendix E.

7.3.3 Rules, facts, actions in the design case

The design case was studied and each of the rules, facts and actions were extracted and formed in a separate list. These lists were then used as a basis for population (coding) of the design agents.

The causal relationships are documented in Appendix F. Each rule is tagged with the engineer who provided the knowledge.

The facts (and potential facts) identified in the design case are documented in Appendix G. Potential facts are statements by the engineers about which they are uncertain, for example, *Toluene has a vapour pressure of 3 bar*. This fact was asserted in order to progress the design, even though the right reference material was not available to tell us that this was fact. Potential facts have been listed to gain an understanding of what needs to be resolved. The design utilised these 'uncertain' facts in order to progress and explore alternatives. In a real life situation the engineer would have to document these assumptions and confirm them at a later date after analysis of the relevant literature.

Appendix H documents the actions identified from the design case. These actions are normally the result of a causal rule (i.e. if condition X then action Y). They were separated from the rules as the design case did not always justify an action (i.e. no conditional X), but was just put forward as an alternative for review. This is of course quite useful where an agent is aware of an alternative but does not have any knowledge regarding when to use it or its associated design issues. The action list also includes our 'design requirements' which do not have justifications (i.e. if we request the computer to design X we do not want to have to justify why we want it to design it!).

7.3.4 Codifying the knowledge in the design case

When the knowledge has been identified and split between the responsible engineers, an agent can be defined for each engineer involved in the discussion. The interests of the agent can be determined in each of the design categories with reference to the appropriate design object (e.g. INTEREST roger DESIGN SYNTHESIS STORAGE or INTEREST Jeremy EVALUATE SELECTION STORAGE). The appropriate response functions (e.g.

put forward a design for X, evaluate the design X for requirement Y) can then be written for each of the agent interests. The simple grammar to enable this communication is documented in section 6.3. These response functions can of course be wrappers to external programs that can provide the solution (e.g. selection programs for DESIGN SELECTION, a CAD database/interface for DESIGN SYNTHESIS). Of course where a wrapper is involved one still has the requirement for design evaluation based upon the agents objectives. This may be provided by the wrapper - externally to the program making the proposal - or by the external package itself.

7.3.5 Inferring rules from other design statements

During the development of the design case, some additional rules were required in order to ensure that both sides of the story were maintained. These additions were due to the information that was inferred but not mentioned during the scenario discussion. Assumptions were made based on the rules, and these appear to be logical assumptions in the context of the discussion. For example:

rule:	a valve on the base of the tank is good for maintenance as the waste can be removed.
assume:	if solids can build up on the base of the tank, then the tank is difficult to maintain without a valve.

rule:	a costly system is required to get the tanker to discharge to pressure vessel.
assume:	it is cheaper to get the tanker to discharge to an atmospheric tank.

rule:	if the fluid is highly volatile and toxic, then a floating roof tank is not appropriate
assume:	atmospheric tanks and pressure vessels are ok for toxic fluids (no weak seals that could potentially leak to atmosphere).

Making these assumptions was necessary in order to avoid the agents reviewing the designs from only a pessimistic view. In some cases a particular design path was considered by the engineers not because they considered it necessarily to be the best solution, but because there were problems associated with the other solutions. In CDEX, the framework will progress design only on the proposals where assessments from the agents are available, and therefore the framework often required the reverse side to any argument to be recorded.

7.4. Result analysis

7.4.1 Format of test case results

Generated text hierarchy

CDEX generates a text file, detailing a design in a hierarchical format that was developed cooperatively by two agents in the CDEX framework. The initial design requirement for the design case presented to the framework was a requirement to feed 10 tonnes of product - currently located off site - to a distillation column. This was presented to CDEX by the agent 'Roger' and refined by both agents until detailed design was complete. The requirement is shown in the hierarchy at the top of the design tree. The format of the hierarchy generated as a result of the design study is provided below. An understanding of this format is necessary in order to understand the results generated by CDEX, and also aids in understanding the structures shown in appendices I and J.

please note: the format for the hierarchy was developed to ease interpretation. However, because of the style of hierarchy (delimiters used etc) it is not amenable to describing using BNF notation. The existence of attributes is therefore covered in the textual description of the attribute, and the structure - with an understanding of the design objects (refinements, object hierarchies etc) - should be quite straight forward.

Design objects

```
[<OBJECT TYPE> name
 {<attribute of design object> + <value>}

 (synthesisedRefinement%nnnn
 ...
 )
 (parametisedRefinement%nnnn
 ...
 )
 (selectedRefinement%nnnn
 ...
 )
]
```

The design objects shown in the hierarchy are those objects that have been accepted as part of the final design in CDEX.

<OBJECT TYPE>	The type of design object
<attribute of design object>	A slot/attribute/value associated with an object that has a value associated with it. The slot may have been proposed for the object itself, or accepted for a more detailed design selection or parametric

design and is a property that is local to the object in question. CDEX propagates up the attributes derived for a more detailed refinement so that the attributes are available to other design processes. A value of a slot listed here therefore may not have been proposed for the design object itself, but for a more specific instance of the object.

The three refinements listed with the object are more detailed design proposals for the object in question. The structure of the synthesisedRefinement, selectedRefinement, and parametisedRefinement is the same and is therefore described below simply as refinement.

Refinement Objects

```
(xxxxxxxxxRefinement%nnnnn
 >>> <proposal>, <by agent>: <objects in proposal>
 > evaluation: <evaluation reference>, <evaluation agent>: <rationale>
 >        pref: <preference> max: <max preference> rev: <rev pref> conf: <conf>
 >        pv: <preference vector>  uv: <utility vector>

 > problem: <conflict reference>, <conflict agent>, <keyword> <constraint type>
 >        <rationale for conflict>
 >        pref: <preference> max: <max preference> rev: <rev pref> conf: <conf>
 >        pv: <preference vector>  uv: <utility vector>

 <> CR strategy:          {       (<cr strategy>, <applied to proposal>) |
                                  (<proposalA> as alternative to <proposalB>)
                          }
 > accepted <proposalAccepted> <
 > Proposal NOT accepted

 {[OBJECT TYPE...
   ...
   ]
 }
)
```

<proposal> unique proposal identifiers (inc. conflicts & evaluations)

<evaluation reference>
<conflict reference>

<by agent> agent responsible for the proposal
<conflict agent>
<evaluation agent>

<objects in proposal>	Objects that are part of the proposal (i.e. refined design)
<rationale> <rationale for conflict>	Reason behind the proposal
<preference>	A weighting denoting how good the proposal is viewed by the agent presenting the proposal/evaluation/conflict.
<max preference>	The maximum potential preference that could be attained by the agent. If the agent only considered a few of his values, then the max preference denotes the highest attainable preference considering these goals.
<rev pref>	A revised preference to give an idea of how good the proposal is from an agents point of view, considering the maximum attainable preference the agent could consider.
<conf>	The confidence an agent has in the proposal (1 is high)
<preference vector>	A vector that depicts how much each of the agents goals were attained. A nil indicates that the goal was not considered.
<utility vector>	The agents personal utility vector - the agents preferences or 'values'. This vector does not change during the design process.
<keyword>	Usually a single word that depicts the type of problem. e.g. for a pipeline potential problems could be MORE FLOW, LESS FLOW, NO FLOW etc.
<constraint type>	HARD or SOFT. Hard indicates the conflict is one that cannot be broken. A soft conflict can be compromised.
<CR strategy>	A specific instance of a conflict resolution strategy. Instances are maintained so that they can keep histories of the conflict for a specific design

refinement.

<applied to proposal>	The proposal to which the conflict resolution strategy was applied.
<proposalA> <proposalB>	Proposal A was proposed as an alternative to proposal B.
<proposalAccepted>	The proposal selected as a result of conflict resolution, or there was no disagreement over the best proposal.

If a proposal was accepted, each object in the proposal (<objects in proposal>) will be shown in the tree and any further design effort is shown.

Example refinement showing proposal nitroParamProp%gen7365 by agent roger. The proposal contains one design object - a storage object (storage%gen7366) together with its associated preferences, confidence and utility vectors. A conflict resolution strategy was applied (consensus) to the proposal shown, and was accepted. The storage object to define in more detail is shown below the proposal.

```
>>> nitroParamProp%gen7365, roger:  storage%gen7366,
> evaluation: gen7458, roger: Nitrogen is available, prefer works
>            pref:0.27  max:0.30  rev:0.90  conf:1
>            pv:nil 0.90 nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
<> CR strategy:  (consensus%gen7481, MAIN::nitroParamProp%gen7358) <>
> accepted nitroParamProp%gen7358 <
[STORAGE storage%gen7359
]
```

Example conflict proposal showing the conflict proposal (conflict%gen86) from agent roger, the problem keyword is NO-PIPE and it is a hard conflict (cannot be broken). The rationale behind the conflict and the associated preferences and utility vectors are shown.

```
> problem: conflict%gen86, roger, NO-PIPE  HARD
>          Not connected to pipeline
>            pref:0.10  max:0.25  rev:0.40  conf:1.00
>            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
```

Ommissions in the design text hierachy

Various omissions are made in the number of proposals put forward for a particular design. These proposals are those proposals that have been put forward as a result of conflict resolution and therefore are not different in content (in these test cases - although potentially they can be) but have different utility vectors applied to the analysis. They are not included here to avoid clutter, although the complete developed scenario can be found

in Appendix I. Proposals have been included in the this section if they are the first proposals put forward by the agent, or the proposal that was accepted as part of the final design. Proposals important to the design are highlighted in bold.

Design 'trace' file

Throughout the design case scenarios you will find references to a 'trace' file. This file contains an order list of events for all the operations on design objects, conflicts, evaluations, proposals etc, and therefore provides an accurate picture of what happened in the design process. This file for the first design case generated over 30,000 lines of text and therefore was not appropriate to include in the thesis. However, where appropriate, various sections have been included in the results to aid understanding. The sections shown are simplified and have had the various object references and irrelevant detail to the test removed.

The elements of the trace file refer to the 'module' in which the message was generated (e.g. Negotiation, Roger etc), and normally a reference to the design object is indicated. Information regarding any methods referred to in the trace file can be found in the detailed design, although the method names are descriptive and need little interpretation. The trace file should be self explanatory once an understanding of the system is attained.

The following discussion on the design scenarios covers each design object accepted by the CDEX system. Each object may in turn be described under the headings 'Attribute', 'Synthesis', 'Parametric', 'Selection', 'Model' and 'Discussion'. The 'Attributes' describes the values pertinent to the object in question that have been accepted. The 'Synthesis', 'Parametric' and 'Selection' headings refer to the detailed design refinements considered and accepted in the CDEX system. The contents of these sections was generated by CDEX and a description of this format is provided in section 7.4.1. The 'Model' is a pictorial representation of the accepted design described under the 'Synthesis' and 'Selection' headings. The 'Discussion' provides a review of why the design shown was accepted and the issues relating to any discrepancy between the CDEX generated model and the model covered in the design meeting.

7.4.2 Best design using case scenario knowledge - Design case I

This case scenario was executed with the following global design criteria:

- concurrent engineering factor = 1.0

This high factor indicates that the system is attempting to find the best solution from the global perspective. If the selected design path does not look promising, there is no penalty in dropping the design and exploring another design path. A low concurrent engineering (CE) factor would account for a certain degree of time spent (depending on the CE factor)

and therefore a particular design path is promoted if design effort has already been spent on exploring a solution for that path. (see section 6.4.4 for details on this factor).

- user criterion for weighting solution:
  confidence = 0.2
  potential-to-succeed = 0.3
  preference = 0.4
  time = 0.1

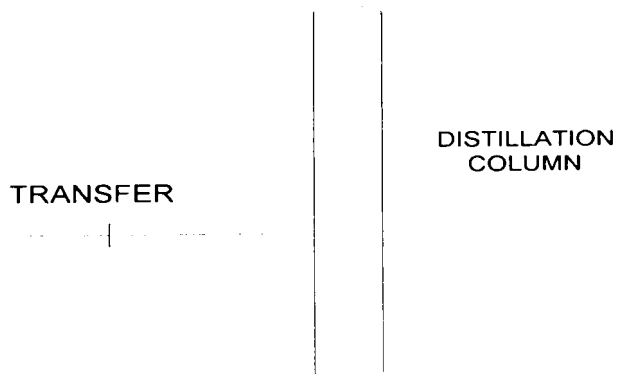
This criterion was used for accessing the weight of which proposal and resolution strategy is most appropriate for resolving the problem. For more information on these factors, refer to section 6.4. This criterion will consider the preference of the proposal most important - i.e. go with the strategy where the result is the most preferred. Second to this the criterion will weight highly a strategy which is likely to resolve the conflict (potential to succeed). The confidence and time are given little consideration but are not so important. It is important however to give them a little consideration to prevent problems. If time is discounted for example, CDEX may spend more time than is necessary with a fruitless strategy. If the strategy is always poor, then the time factor will eventually lead to the strategy being a poor choice and therefore not selected.

TRANSFER transfer

The initial requirement was presented to the CDEX framework for design by roger. No detailed design was performed on the still, the transfer mechanism however was assessed in considerable detail.

Attributes:
outlet-mass-flow-rate 10; unit-process-downstream MAIN::still; unit-process-upstream offsite;

Model:

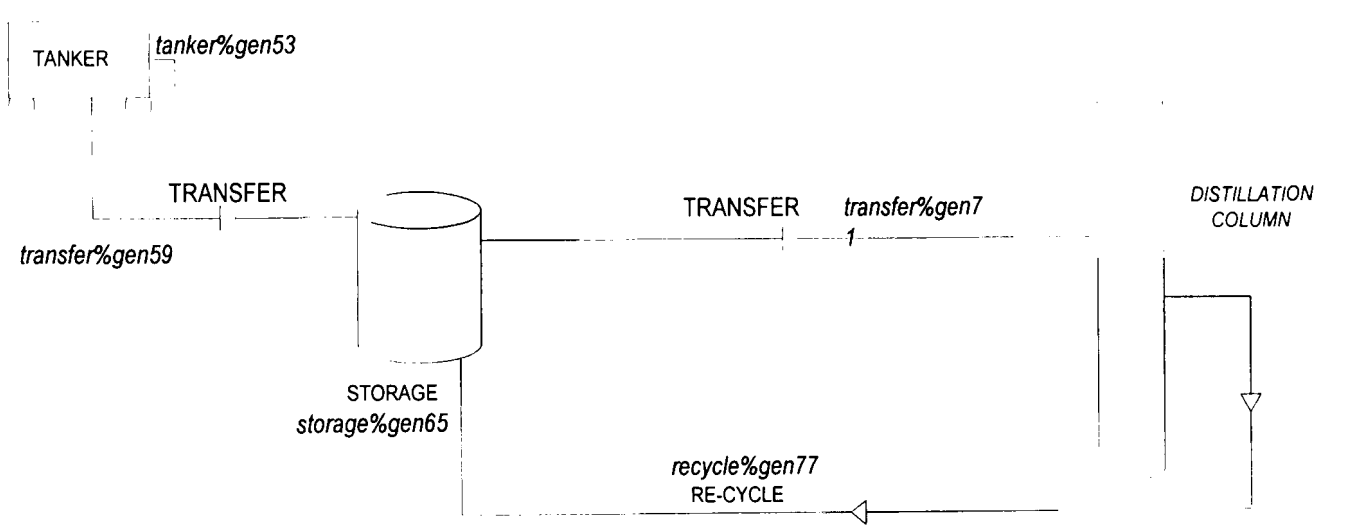


Synthesis:

```
>>> synthProp%gen32, roger:  stream%gen33,
> problem: conflict%gen86, roger, NO-PIPE  HARD
```

```
>        Not connected to pipeline
>            pref:0.10  max:0.25  rev:0.40  conf:1.00
>            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10

>>> synthProp%gen39, roger:  tanker%gen40, stream%gen46,
> evaluation: gen87, roger: Lots of  traffic, low connection time
>            pref:0.29  max:0.60  rev:0.48  conf:1
>            pv:nil 0.60 0.36 nil nil  uv:0.25 0.30 0.30 0.05 0.10

>>> synthProp%gen52, roger:  tanker%gen53, transfer%gen59, storage%gen65,
                            transfer%gen71, recycle%gen77,
> evaluation: gen88, roger: Buy stock when cheap
>            pref:0.19  max:0.25  rev:0.76  conf:0.5
>            pv:0.76 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen91, jeremy, HIGH-STOCK MORE-FLOW  SOFT
>        best not to hold stocksLoose level from still - 6-8m3 vapour
>            pref:0.25  max:0.40  rev:0.63  conf:0.80
>            pv:nil nil 0.63 nil nil  uv:0.20 0.30 0.40 0.00 0.10
 > accepted synthProp%gen52 <
```

Model:



Discussion:

There was no conflict over the best proposal in this case. Both Roger and Jeremy agreed on the proposal where the toluene was tankered in and stored before being transferred to the distillation column.

TANKER tanker%gen53

No further design.

Discussion:

Where no further design is indicated, this means that the agents involved in the design have not presented proposals for the object in question. In this case, neither agent wished to

elaborate on the design of a tanker. Obviously if design is complete, then this is indicated by 'No further design' being associated with a design object.

TRANSFER transfer%59

Synthesis:

>>> transFromTank%gen138, roger: storage%gen151, stream%gen145,
tripValve%gen139,
> evaluation: gen244, roger: Control valve prevents pressure blow through.
> pref:0.26 max:0.30 rev:0.88 conf:0.8
> pv:nil nil 0.88 nil nil uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen250, jeremy, MORE_FLOW SOFT
> Trip valve may jam open
> pref:0.30 max:0.40 rev:0.76 conf:0.80
> pv:nil nil 0.76 nil nil uv:0.20 0.30 0.40 0.00 0.10

>>> transFromTank%gen157, roger: stream%gen158, stream%gen164, pump%gen170,
> evaluation: gen245, roger: Pumping is apropriate
> pref:0.00 max:0.00 rev:0.00 conf:0.8
> pv:nil nil nil nil nil uv:0.25 0.30 0.30 0.05 0.10

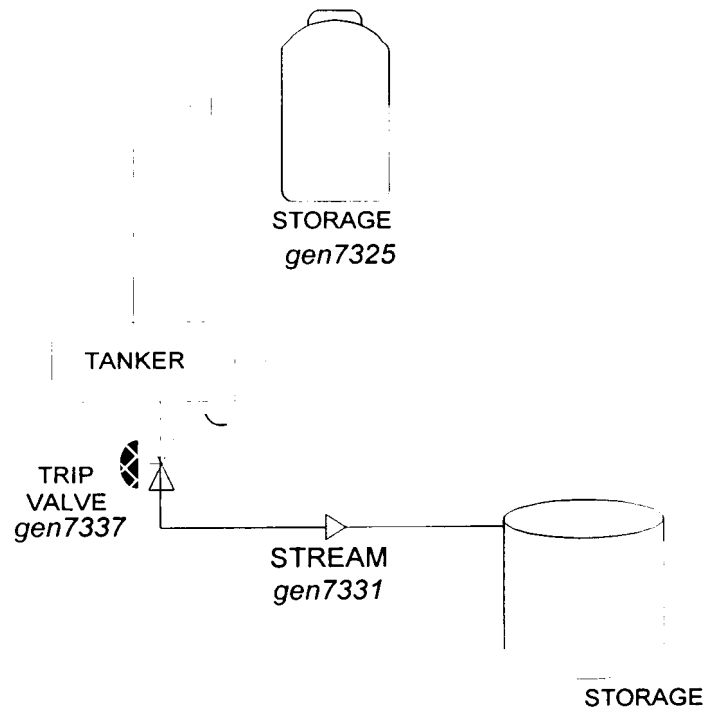
>>> transFromTank%gen190, jeremy: gasStorage%gen191, stream%gen197,
> evaluation: gen247, roger: Control valve needed to prevent pressure blow through.
> pref:0.16 max:0.30 rev:0.52 conf:0.8
> pv:nil nil 0.52 nil nil uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen253, jeremy: Pressure through transfer when tank empty
> pref:0.34 max:0.40 rev:0.84 conf:1
> pv:nil nil 0.84 nil nil uv:0.20 0.30 0.40 0.00 0.10
.....
.....
.....

>>> gen7324, roger: gen7325, gen7331, gen7337,
> evaluation: gen7344, roger: Control valve prevents pressure blow through.
> pref:0.27 max:0.30 rev:0.88 conf:0.8
> pv:nil nil 0.88 nil nil uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen7345, jeremy, MORE_FLOW SOFT
> Trip valve may jam open
> pref:0.23 max:0.30 rev:0.76 conf:0.80
> pv:nil nil 0.76 nil nil uv:0.25 0.30 0.30 0.05 0.10
<> CR strategy: (compromise%gen259, MAIN::transFromTank%gen138)
(MAIN::gen292 as alternative to MAIN::transFromTank%gen138)
(consensus%gen264, MAIN::transFromTank%gen138)
(compromise%gen259, MAIN::gen292)
(MAIN::gen660 as alternative to MAIN::gen292)
(consensus%gen264, MAIN::gen660)
(compromise%gen259, MAIN::gen660)
(MAIN::gen1313 as alternative to MAIN::gen660)
(consensus%gen264, MAIN::gen1313)
(compromise%gen259, MAIN::gen1313)
(MAIN::gen2350 as alternative to MAIN::gen1313)

```
                (consensus%gen264, MAIN::gen2350)
                (compromise%gen259, MAIN::gen2350)
                (MAIN::gen3237 as alternative to MAIN::gen2350)
                (consensus%gen264, MAIN::gen3237)
                (compromise%gen259, MAIN::gen3237)
                (MAIN::gen3878 as alternative to MAIN::gen3237)
                (consensus%gen264, MAIN::gen3878)
                (majorityRule%gen265, MAIN::transFromTank%gen138)
                (majorityRule%gen265, MAIN::gen2350)
                (majorityRule%gen265, MAIN::gen3237)
                (majorityRule%gen265, MAIN::gen3878)
                (compromise%gen259, MAIN::gen3878)
                (MAIN::gen6244 as alternative to MAIN::gen3878)
                (consensus%gen264, MAIN::gen6244)
                (majorityRule%gen265, MAIN::gen6244)
                (majorityRule%gen265, MAIN::gen292)
                (majorityRule%gen265, MAIN::gen660)
                (compromise%gen259, MAIN::gen6244)
                (MAIN::gen7324 as alternative to MAIN::gen6244)
                (consensus%gen264, MAIN::gen7324)
                (majorityRule%gen265, MAIN::gen7324) <>
    > accepted gen7324 <
```

Model:

System to enable 'blowthrough' - where the tank contents is blown out by applying pressure on the surface of the fluid in the tank:



Discussion:

Three proposals are put forward for unloading the tank, one solution is to pump it out. The other solution is to 'blow through'. Two proposals are put foreword for the 'blow through' solution - one includes a trip valve for preventing blow through when the tanker is empty, creating a high pressure in the storage. Both Jeremy and Roger prefer the solution to blow

through, although Jeremy prefers the solution without a trip valve. whereas Roger prefers the trip valve. Jeremy thinks that the control valve may jam open which is a bigger problem than there being no trip valve. It is difficult to interpret this result from analysis of the discussion - possibly if one relies on the trip valve for preventing flow when the tanker is empty then this is a rationale conclusion from the system. However they eventually agree on the approach considered in the design case.

The interesting point to note in this automated design, is the number of strategies applied to resolving conflict. The following is an ordered list of the strategies applied:

> compromise, consensus, compromise, consensus, compromise, consensus, compromise, consensus, compromise, consensus, compromise, consensus, majorityRule, majorityRule, majorityRule, majorityRule, compromise, consensus, majorityRule, majorityRule, majorityRule, compromise, consensus, majorityRule.

Initially the conflict is identified, and a compromise strategy is deemed the most appropriate solution. The following shows that proposals 138 and 190 were determined to be the best proposals and therefore conflict exists (the agents disagree on the best approach). The trace file below then shows that the best solution could be attained by compromising on proposal 138, which is the first strategy to be applied to resolve the conflict.

....
negotiation: get-agent-best-proposals: final best list:
roger [MAIN::transFromTank%gen138] 0.88 jeremy [MAIN::transFromTank%gen190] 0.84
negotiation: get-best-proposals: [MAIN::transFromTank%gen138] [MAIN::transFromTank%gen190]
negotiation: A conflict exists, no best proposal could be found.
negotiation: Conflict identified
negotiation: Determining best strategy
negotiation: compromise proposal transFromTank%gen138
negotiation: compromise proposal transFromTank%gen157
negotiation: compromise: only single view problem
negotiation: compromise proposal transFromTank%gen190
....
negotiation: determine-best-strategy: ** strategy **[compromise%gen259]
negotiation: determine-best-strategy: Proposal:: [MAIN::transFromTank%gen138]
negotiation: determine-best-strategy: confidence 0.8
negotiation: determine-best-strategy: potential 0.7090909090909091
negotiation: determine-best-strategy: preference 0.8200000000000001
negotiation: determine-best-strategy: time 0.98
negotiation: determine-best-strategy: * total weight * 0.7987272727272728
negotiation: determine-best-strategy: Proposal:: [MAIN::transFromTank%gen190]
negotiation: determine-best-strategy: confidence 0.9
negotiation: determine-best-strategy: potential 0.6357142857142857
negotiation: determine-best-strategy: preference 0.6799999999999999
negotiation: determine-best-strategy: time 0.98
negotiation: determine-best-strategy: * total weight * 0.7407142857142857
negotiation: determine-best-strategy: ** strategy **[abandonGoals%gen260]
negotiation: determine-best-strategy: ** strategy **[generateAlternative%gen261]
.....

The agents respond with their new compromise proposals/reviews, and an agreement is found by applying a consensus strategy. When further, more detailed design is performed on the selected proposal (gen138), the design does not progress as good as initially expected, and when the more detailed design requests for further design effort (permission-to-proceed) - the request is denied. When the request is denied. the negotiation mechanism re-evaluated all the options and determined that the path was again the most suitable path. and design then progressed along the previous design path. This iteration was due to a poor evaluation of how design was expected to turn out by the agents in the early design phase. This iteration obviously leads to many conflicts and therefore many different strategies are ultimately applied to resolve all the conflicts. identified. This was the case in resolving this particular part of the design.

STORAGE gen7325

Attributes:
..location offsite; fluid nitrogen; reliability 100; unit-process-downstream MAIN::tanker%gen53;

Parametric design:

>>> nitroParamProp%gen7358, roger: storage%gen7359,
> evaluation: gen7454, roger: Nitrogen is available, prefer works
> pref:0.24 max:0.30 rev:0.80 conf:1
> pv:nil 0.80 nil nil nil uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen7459, jeremy, NOT_POSSIBLE HARD
> Nitrogen supply not available on tanker
> pref:0.15 max:0.30 rev:0.50 conf:0.50
> pv:nil 0.50 nil nil nil uv:0.20 0.30 0.40 0.00 0.10

>>> nitroParamProp%gen7365, roger: storage%gen7366,
> evaluation: gen7458, roger: Nitrogen is available, prefer works
> pref:0.27 max:0.30 rev:0.90 conf:1
> pv:nil 0.90 nil nil nil uv:0.25 0.30 0.30 0.05 0.10
<> CR strategy: (consensus%gen7481, MAIN::nitroParamProp%gen7358) <>
> accepted nitroParamProp%gen7358 <

Discussion:

Both an onsite and offsite nitrogen supply was proposed. The offsite supply (available on the tanker) was selected (see attributes). There is a HARD conflict associated with the proposal accepted as jeremy thought that a supply was not available on the tanker, although he was not sure. If nitrogen was available the solution was ok. There is therefore a low confidence in the possibility that a nitrogen supply did not exist.

Synthesis:

```
>>> heatedTank%gen7503, jeremy:  storage%gen7504, coil%gen7511,
> evaluation: gen7530, jeremy: Good for maintenace
>            pref:0.10  max:0.10  rev:1.00  conf:1
>            pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
> problem: conflict%gen7528, roger, NOT-REQUIRED  HARD
>        Heating coil not required
>            pref:0.10  max:0.25  rev:0.40  conf:1.00
>            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> Proposal was NOT accepted
```

Discussion:

A proposal was put forward to design the nitrogen supply tank to have a heating coil. This knowledge presented in the design case with the main storage supply to the still itself. Due to the knowledge being specified as important in the design of a storage mechanism, it therefore applies itself to the nitrogen supply as well. If the proposal put forward is not appropriate then the defined knowledge is not explicit enough regarding the design context. In this case the proposal was not accepted because of a hard conflict proposed by roger, heating coils are just not required.

STREAM gen7331

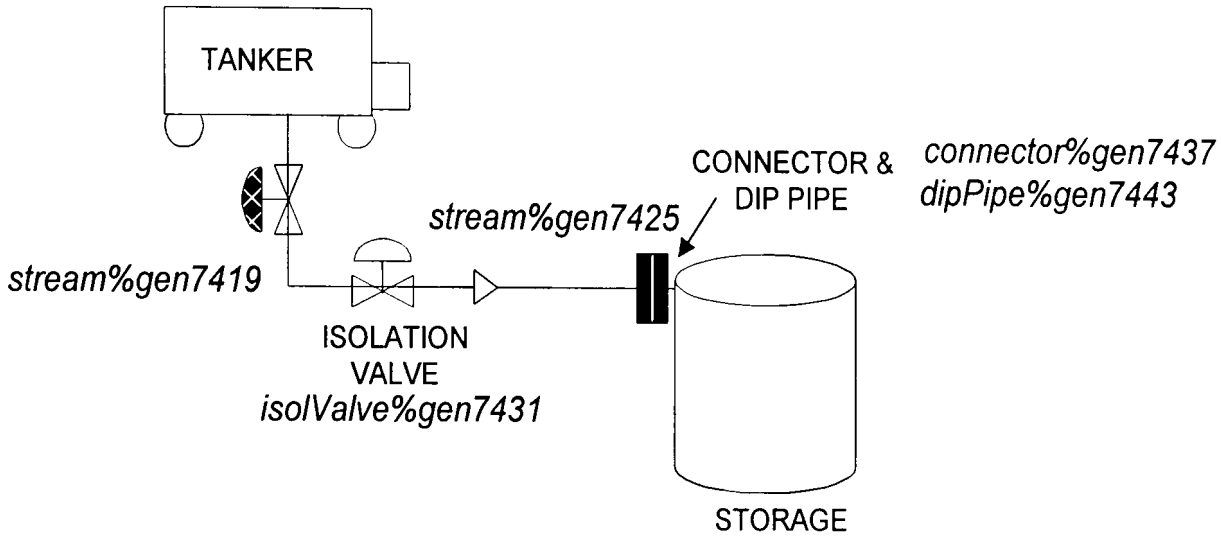
Attributes:
..unit-process-downstream MAIN:: storage%gen65; unit-process-upstream MAIN ::tripValve %gen139;

Synthesis:

```
>>> streamToTank%gen7374, roger:  stream%gen7375, stream%gen7381,
                                  isolValve%gen7387,
> evaluation: gen7466, roger: Connector flow in environment;Reverse flow from
                                  connector; Potential for splash filling;Control valve shut;
>            pref:0.11  max:0.30  rev:0.36  conf:0.7
>            pv:nil nil 0.36 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen7469, jeremy: Iso. valve ok for maint, frothing potential
>            pref:0.48  max:0.80  rev:0.60  conf:1
>            pv:nil 0.20 0.80 nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

>>> streamToTank%gen7393, roger:  stream%gen7394, stream%gen7400,
                                     isolValve%gen7406, connector%gen7412,
> evaluation: gen7467, roger: Potential for splash filling;Control valve shut;
>            pref:0.16  max:0.30  rev:0.52  conf:0.8
>            pv:nil nil 0.52 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen7471, jeremy: Iso. valve ok for maint, frothing potential
>            pref:0.48  max:0.80  rev:0.60  conf:1
>            pv:nil 0.20 0.80 nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

>>> transFromTank%gen7418, jeremy:  stream%gen7419, stream%gen7425,
          isolValve%gen7431, connector%gen7437, dipPipe%gen7443,
```

> evaluation: gen7468, roger: Control valve shut;
> pref:0.26 max:0.30 rev:0.88 conf:0.8
> pv:nil nil 0.88 nil nil uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen7473, jeremy: Iso. Valve good for maint, dip pipe prevents static
> pref:0.50 max:0.50 rev:1.00 conf:1
> pv:nil nil 1.00 nil 1.00 uv:0.20 0.30 0.40 0.00 0.10
> accepted transFromTank%gen7418 <

Model:



Discussion:

Both Roger and Jeremy agreed on the same proposal, which was the approach covered in the real life design case.

STREAM stream%gen7419

No further design performed.

STREAM stream%gen7425

No further design performed.

ISOLATION_VALVE isolValve%gen7431

No further design performed.

CONNECTOR connector%gen7437

No further design performed.

DIP_PIPE dipPipe%gen7443

No further design performed.

CONTROL_VALVE gen7337

No further design performed.

STORAGE storage%gen65

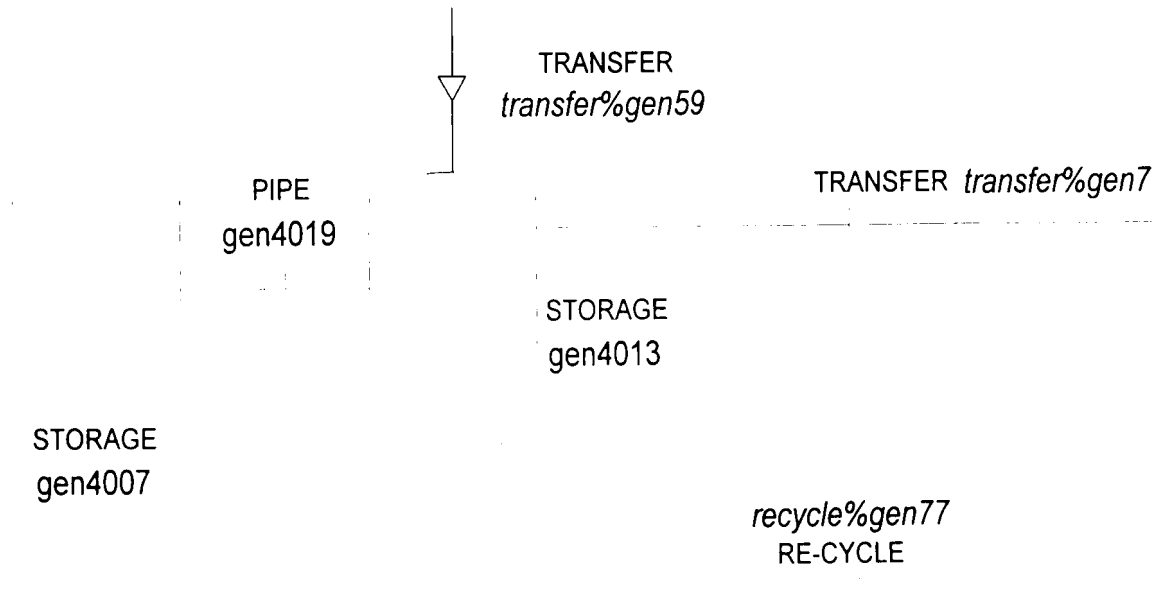
Attributes:
..reliability 90; unit-process-downstream MAIN::transfer%gen71; unit-process-upstream MAIN ::transfer%gen59

Synthesis:

```
>>> storageFarm%gen177, roger:  storage%gen178, storage%gen184,
> evaluation: gen246, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen252, jeremy: Re-cycle without flow control problems, control expensive,
                                >1 tank, ok for maintenance
>           pref:0.33  max:0.60  rev:0.55  conf:1
>           pv:0.40 0.50 nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

>>> heatedTank%gen203, jeremy:  storage%gen204, coil%gen211,
> evaluation: gen255, jeremy: Poor for maintenance
>           pref:0.02  max:0.10  rev:0.20  conf:1
>           pv:nil nil nil nil 0.20  uv:0.20 0.30 0.40 0.00 0.10
> problem: conflict%gen248, roger, NOT-REQUIRED  HARD
>       Heating coil not required
>           pref:0.10  max:0.25  rev:0.40  conf:1.00
>           pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10

>>> storageFarm%gen217, jeremy:  storage%gen218, storage%gen224,
                                    pipe%gen230,
> evaluation: gen249, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen257, jeremy: Re-cycle to single tank does not split, >1 tank
                                good for maintenance.
>           pref:0.35  max:0.60  rev:0.58  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
  .........
  .........
  .........
>>> gen4006, jeremy:  gen4007, gen4013, gen4019,
> evaluation: gen4304, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10
> evaluation: gen4306, jeremy: Re-cycle to single tank does not split, >1
                                tank good for maintenance.
>           pref:0.35  max:0.60  rev:0.58  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10
```

.....
.....
.....
<> CR strategy: (smoothing%gen279, MAIN::storageFarm%gen217)
(smoothing%gen279, MAIN::storageFarm%gen177)
(compromise%gen275, MAIN::storageFarm%gen217)
(MAIN::gen965 as alternative to MAIN::storageFarm%gen217)
(smoothing%gen279, MAIN::gen965)
(consensus%gen280, MAIN::gen965)
(compromise%gen275, MAIN::gen965)
(MAIN::gen2579 as alternative to MAIN::gen965)
(consensus%gen280, MAIN::gen2579)
(compromise%gen275, MAIN::gen2579)
(MAIN::gen3139 as alternative to MAIN::gen2579)
(consensus%gen280, MAIN::gen3139)
(compromise%gen275, MAIN::gen3139)
(MAIN::gen3462 as alternative to MAIN::gen3139)
(consensus%gen280, MAIN::gen3462)
(smoothing%gen279, MAIN::gen2579)
(compromise%gen275, MAIN::gen3462)
(MAIN::gen4006 as alternative to MAIN::gen3462)
(consensus%gen280, MAIN::gen4006)
(smoothing%gen279, MAIN::gen3139)
(majorityRule%gen281, MAIN::gen965)
(majorityRule%gen281, MAIN::gen2579)
(compromise%gen275, MAIN::gen4006)
(MAIN::gen6125 as alternative to MAIN::gen4006)
(consensus%gen280, MAIN::gen6125)
(majorityRule%gen281, MAIN::gen3139)
(majorityRule%gen281, MAIN::gen3462)
(majorityRule%gen281, MAIN::gen4006) <>
> accepted gen4006 <

Model:



Discussion:

Conflict was identified in this case between proposals 177 (two tanks, recycle to both) and 217 (two tanks, recycle to one). This conflict is shown in the following design trace. The smoothing strategy was applied in this case to bring more information into the design review. Smoothing is an computationally expensive strategy in that more design has to take place in order for the design approach to be assessed. In this scenario however quality has been shown to be high (i.e. high preference is important), and therefore smoothing is applied.

```
get-agent-best-proposals: best list: (roger [MAIN::storageFarm%gen177] 1.0 jeremy
[MAIN::storageFarm%gen217] 0.5833333333333334)
negotiation: get-agent-best-proposals: final best list: roger [MAIN::storageFarm%gen177] 1.0
jeremy [MAIN::storageFarm%gen217] 0.5833333333333334
negotiation: get-best-proposals: [MAIN::storageFarm%gen177] [MAIN::storageFarm%gen217]
negotiation: A conflict exists, no best proposal could be found.
negotiation: Conflict identified
negotiation: Determining best strategy
negotiation: compromise proposal storageFarm%gen177
....
negotiation: determine-best-strategy: ** strategy **[smoothing%gen279]
negotiation: determine-best-strategy: Proposal:: [MAIN::storageFarm%gen177]
negotiation: determine-best-strategy: confidence 1.0
negotiation: determine-best-strategy: potential 0.7
negotiation: determine-best-strategy: preference 0.775
negotiation: determine-best-strategy: time 0.95
negotiation: determine-best-strategy: * total weight * 0.8150000000000001
negotiation: determine-best-strategy: Proposal:: [MAIN::storageFarm%gen217]
negotiation: determine-best-strategy: confidence 1.0
negotiation: determine-best-strategy: potential 0.7
negotiation: determine-best-strategy: preference 0.7916666666666667
negotiation: determine-best-strategy: time 0.95
negotiation: determine-best-strategy: * total weight * 0.8216666666666668
....
negotiation: best strategy is smoothing%gen279 weight 0.821667
negotiation: Permission to proceed with strategyEvaluation%gen286, effort expended 0
negotiation: Permission to proceed granted synthesisedRefinement%gen5
negotiation: applying strategy smoothing%gen279
proposal: Proposal storageFarm%gen217 developed
....
```

The conflict is initially resolved after the fifth strategy applied. Initially the smoothing approach is applied twice to bring more information into the conflict scenario, a solution is then compromised, and then further information is obtain (smoothing) to support the design path. Consensus is then applied and a proposal is chosen. As design progresses, design does not proceed as expected, and again the alternatives are reviewed and conflict resolution is applied to resolve the conflict given the new more informed criteria gained from further insight into the design path. This iteration results in the long list of conflict resolution strategies applied as shown above.

STORAGE gen4007

Attributes:
..material-of-construction mildsteel; reliability 99; unit-process-downstream MAIN::transfer%gen71; unit-process-upstream MAIN::transfer%gen59;

Synthesis:

```
>>> heatedTank%gen4756, jeremy:  storage%gen4757, coil%gen4763,
> evaluation: gen5061, jeremy: Good for maintenace
>            pref:0.10  max:0.10  rev:1.00  conf:1
>            pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
> problem: conflict%gen5051, roger, NOT-REQUIRED  HARD
>        Heating coil not required
>            pref:0.10  max:0.25  rev:0.40  conf:1.00
>            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> Proposal was NOT accepted
```

Selected:

```
>>> storageSelProp%gen4735, jeremy:  atmosTank%gen4736,
> evaluation: gen5048, roger: Not as expensive as pressure vessel
>            pref:0.25  max:0.25  rev:1.00  conf:1
>            pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen5058, jeremy, NOT_POSSIBLE  HARD
>        Cannot store hot toluene in atmos. tank
>            pref:0.35  max:0.40  rev:0.88  conf:0.30
>            pv:nil nil 0.88 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> storageSelProp%gen4742, jeremy:  floatingRoofTank%gen4743,
> evaluation: gen5049, roger: Not as expensive as pressure vessel
>            pref:0.25  max:0.25  rev:1.00  conf:1
>            pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen5059, jeremy, NOT_POSSIBLE  HARD
>        Highly volatile and toxic fluid, floating roof not appropriate
>            pref:0.00  max:0.00  rev:0.00  conf:1.00
>            pv:nil nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> storageSelProp%gen4749, jeremy:  pressureVessel%gen4750,
> evaluation: gen5050, roger: Pressure vessel expensive
>            pref:0.10  max:0.25  rev:0.40  conf:1
>            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen5060, jeremy: Costly system to get tank to discharge to pressure vessel
>            pref:0.00  max:0.20  rev:0.00  conf:1
>            pv:0.00 nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10
> accepted storageSelProp%gen4735 <
```

Discussion:

Jeremy did not believe that hot toluene could be stored in an atmospheric tank, although

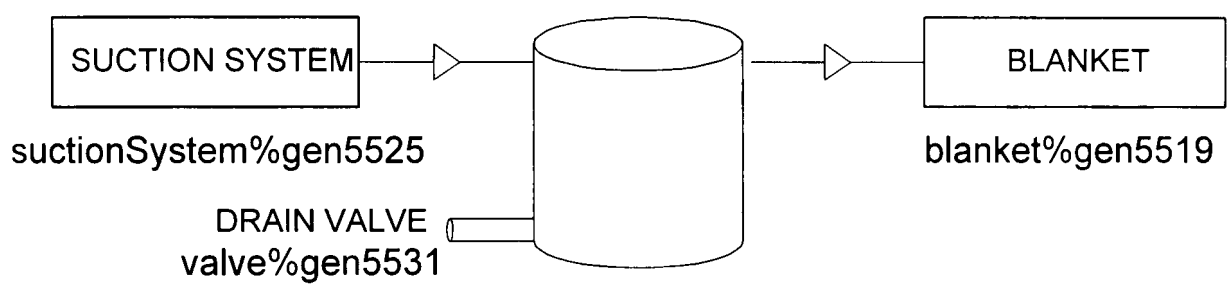
he was unsure of the conditions and placed a low belief on a problem existing. Both agents thought however in their discussions that the atmospheric tank was a good approach - although Jeremy did have reservations on the temperature of the toluene being returned from the still.

ATMOSPHERIC_TANK atmosTank%gen4736

Synthesis:

>>> atmosTankSynth%gen5512, jeremy: atmosTank%gen5513, blanket %gen5519, suctionSystem %gen5525, valve %gen5531,
> evaluation: gen5654, jeremy: Valve good for removing solids; Good for maintenace
> pref:0.10 max:0.10 rev:1.00 conf:1
> pv:nil nil nil nil 1.00 uv:0.20 0.30 0.40 0.00 0.10
> accepted atmosTankSynth%gen5512 <

Model:



Discussion:

Only Jeremy put a proposal forward for the synthesis of the atmospheric tank and only Jeremy put forward an evaluation - a rationale of why the proposal was good. In this case there is no conflict, as no hard conflict has been identified, and there is only one best proposal as only one agent has reviewed the proposal.

ATMOSPHERIC_TANK atmosTank%gen5513

Attributes:

..material-of-construction mildsteel

Parametric:

>>> paramProp%gen5744, roger: storage%gen5746,
> accepted paramProp%gen5744 <

Discussion:

No comments were presented on the parametric design of the atmospheric tank. In this case the proposal was only to indicate that the tank material should be mild steel, as put forward by Roger in the real life design case. No evaluations were put forward for the proposal. No one agreed or disagreed so therefore the proposal was accepted.

BLANKET blanket%gen5519

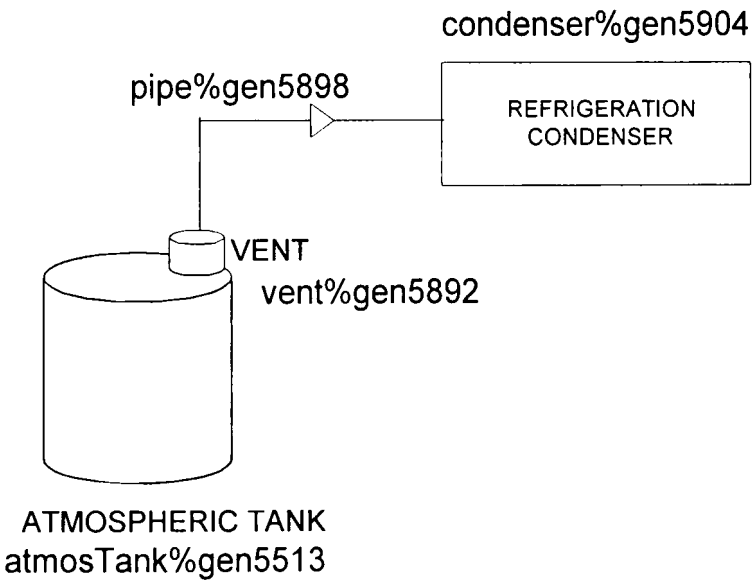
Attributes:
..unit-process-downstream MAIN::atmosTank%gen5513; unit-process-upstream MAIN:: atmosTank %gen5513

Synthesis:

```
>>> blanket%gen5784, roger:  fan%gen5785, stream%gen5791, oxidiser%gen5797,
                                    trip%gen5803,
 > evaluation: gen5985, jeremy: Expensive;No vent with TO;Back pressure problem;
 >          pref:0.33  max:0.60  rev:0.55  conf:0.6
 >          pv:0.30 nil 0.67 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> blanket%gen5822, jeremy:  fan%gen5823, stream%gen5829,
                                        oxidiser%gen5835,
 > evaluation: gen5988, jeremy: Expensive;No vent with TO;Back pressure problem;
 >          pref:0.33  max:0.60  rev:0.55  conf:0.6
 >          pv:0.30 nil 0.67 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> blanket%gen5841, jeremy:  fan%gen5842, stream%gen5848,
                          oxidiser%gen5854, vent%gen5860, trip%gen5866,
 > evaluation: gen5981, roger: Birds can next in the vents.
 >          pref:0.21  max:0.30  rev:0.70  conf:0.3
 >          pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > evaluation: gen5990, jeremy: Expensive;Backpressure with TO resolved
     with vent; Vent open creates flam. atmos.;Back pressure problem;Vent can block;
 >          pref:0.39  max:0.60  rev:0.65  conf:0.6
 >          pv:0.30 nil 0.82 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> blanket%gen5872, jeremy:  vent%gen5873, pipe%gen5879, flare%gen5885,
 > evaluation: gen5982, roger: Birds can next in the vents.
 >          pref:0.21  max:0.30  rev:0.70  conf:0.3
 >          pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > evaluation: gen5992, jeremy: Cheaper;Flair potentially poor solution for future;
                                    Vent can block;
 >          pref:0.52  max:0.60  rev:0.87  conf:0.5
 >          pv:0.80 nil 0.90 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> blanket%gen5891, jeremy:  vent%gen5892, pipe%gen5898,
                                    condenser%gen5904,
> evaluation: gen5983, roger: Birds can next in the vents.
>          pref:0.21  max:0.30  rev:0.70  conf:0.3
```

```
>            pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen5994, jeremy: Cheaper;Vent can block;
>            pref:0.52  max:0.60  rev:0.87  conf:0.6
>            pv:0.80 nil 0.90 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> blanket%gen5910, jeremy:  vent%gen5911, pipe%gen5917,
                        atmosphere%gen5923,
> evaluation: gen5984, roger: Birds can nest in the vents.
>            pref:0.21  max:0.30  rev:0.70  conf:0.3
>            pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen5996, jeremy, NOT_POSSIBLE  HARD
>         Not allowed to vent toluene to atmosphere.
>            pref:0.00  max:0.00  rev:0.00  conf:1.00
>            pv: uv:

>>> gen6106, jeremy:  gen6107, gen6113, gen6119,
> evaluation: gen6167, roger: Birds can nest in the vents.
>            pref:0.26  max:0.37  rev:0.70  conf:0.3
>            pv:nil nil 0.70 nil nil  uv:0.22 0.30 0.37 0.02 0.10
> evaluation: gen6169, jeremy: Cheaper;Vent can block;
>            pref:0.50  max:0.58  rev:0.86  conf:0.6
>            pv:0.80 nil 0.90 nil nil  uv:0.22 0.30 0.37 0.02 0.10
<> CR strategy:    (compromise%gen6055, MAIN::blanket%gen5891)
              (MAIN::gen6106 as alternative to MAIN::blanket%gen5891)
              (consensus%gen6060, MAIN::blanket%gen5891) <>
> accepted blanket%gen5891 <
```

Model:



Discussion:

Roger and Jeremy initially disagree on the best proposal, Jeremy preferring the flared solution, and roger preferring solution to blow through to thermal oxidiser.

negotiation: get-agent-best-proposals: final best list: jeremy [MAIN::blanket%gen5872] 0.86 roger [MAIN::blanket%gen5841] 0.7

negotiation: get-best-proposals: [MAIN::blanket%gen5872] [MAIN::blanket%gen5841]
negotiation: A conflict exists, no best proposal could be found.

A compromise solution was considered, and neither of the agents most preferred proposals identified above were considered for further analysis. The solution that seemed most acceptable to both agents was the solution proposing a condenser (gen5891) and therefore a compromise strategy was applied to achieve agreement on this proposal. The compromised solution was not successful as the compromised difference was too small to get the agents to agree on the best proposal.

The consensus strategy has the same weighting as the compromise strategy, and therefore the first strategy applied could have been either strategy. After the compromise solution failed, it had obviously spent time in resolving conflict (however small) which decreased its chances of being selected a second time. Since initially the compromise solution looked as good as the consensus strategy, the consensus strategy now looked at the more promising solution, and in this case was successful.

The following section of the trace file shows both the compromise and consensus strategies having the same weighting for preference in resolving the conflict:

negotiation: determine-best-strategy: ** **strategy** **[compromise%gen6055]
...
negotiation: determine-best-strategy: Proposal:: [MAIN::blanket%gen5891]
negotiation: determine-best-strategy: confidence 0.45
negotiation: determine-best-strategy: potential 0.6923076923076923
negotiation: determine-best-strategy: preference 0.7833333333333332
negotiation: determine-best-strategy: time 0.98
negotiation: determine-best-strategy: * total weight * **0.709**
...
negotiation: determine-best-strategy: ** **strategy** **[consensus%gen6060]
negotiation: determine-best-strategy: Proposal:: [MAIN::blanket%gen5891]
negotiation: determine-best-strategy: confidence 0.45
negotiation: determine-best-strategy: potential 0.6923076923076923
negotiation: determine-best-strategy: preference 0.7833333333333332
negotiation: determine-best-strategy: time 0.98
negotiation: determine-best-strategy: * total weight * **0.709**

VENT vent%gen5892

No further design performed.

PIPE pipe%gen5898

No further design performed.

REFRIGERATION_CONDENSER condenser%gen5904

No further design performed.

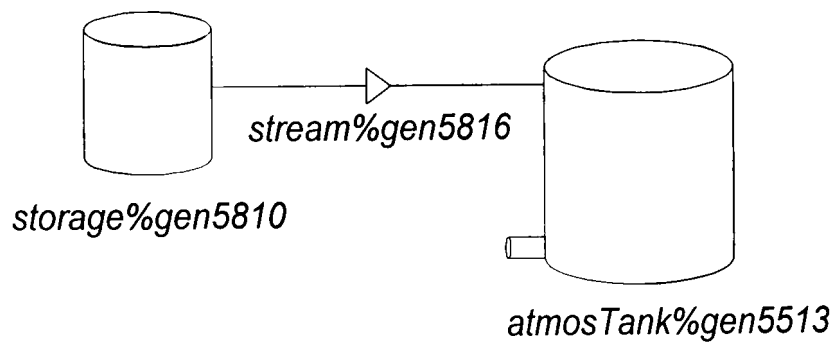
SUCTION_SYSTEM suctionSystem%gen5525

Attributes:
..unit-process-downstream MAIN::atmosTank%gen5513; unit-process-upstream MAIN ::atmosTank%gen5513

Synthesis:

```
>>> suctionProp%gen5809, roger:  storage%gen5810, stream%gen5816,
> problem: conflict%gen5987, jeremy, HIGH-STOCK  SOFT
>        best not to hold stocks
>            pref:0.39  max:0.40  rev:0.96  conf:0.80
>            pv:nil nil 0.96 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> suctionSysProp%gen5929, jeremy:  suctionSystem%gen5930,
> evaluation: gen5998, jeremy: Back breaker blow and create flammable
                                  atmosphere
>            pref:0.25  max:0.40  rev:0.62  conf:1
>            pv:nil nil 0.62 nil nil  uv:0.20 0.30 0.40 0.00 0.10
> accepted suctionProp%gen5809 <
```

Model:



Discussion:

No conflict existed in this case.

STORAGE storage%gen5810

Attributes:
..fluid nitrogen; reliability 100; unit-process-downstream MAIN::stream%gen5816

Synthesis:

```
>>> heatedTank%gen6175, jeremy:  storage%gen6176, coil%gen6183,
> evaluation: gen6272, jeremy: Good for maintenace
>            pref:0.10  max:0.10  rev:1.00  conf:1
>            pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
> problem: conflict%gen6270, roger, NOT-REQUIRED  HARD
>        Heating coil not required
>            pref:0.10  max:0.25  rev:0.40  conf:1.00
>            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> Proposal was NOT accepted
```

Discussion:

The suction system was decomposed into a requirement for a storage of nitrogen which provided a constant supply of nitrogen for the atmospheric tank in low pressure conditions. The knowledge supplied in the design case indicated that for tanks, the need for a heating coil should be reviewed. From the real life design discussion a heating coil was not discussed for the nitrogen supply, although the discussions on the need for heating coils on a tank did not explicitly preclude the nitrogen supply. From this case it is possible to deduce either of the following:

- the information/knowledge defined in the rule base is not explicit enough (does the rule regarding the need for a heating coil need to include some knowledge regarding the fluid?)
- that having a heating coil for a nitrogen storage supply is an interesting question that is not always explored.

The second issue above resents an interesting question. Through defining knowledge at a level appropriate in a hierarchy (e.g. heating coil for storage and not atmospheric tank), it could be considered that when the knowledge is being applied to a nitrogen supply (which also consists of a tank), that this knowledge is being applied through 'analogy'.

In the design case heating coils were not considered important, and in this case the proposal for a heating coil was rejected.

STREAM stream%gen5816

No further design performed.

VALVE valve%gen5531

No further design performed.

STORAGE gen4013

Attributes:

..material-of-construction mildsteel; reliability 99; unit-process-upstream MAIN::pipe%gen230

Synthesis:

```
>>> heatedTank%gen4790, jeremy:  storage%gen4791, coil%gen4797,
> evaluation: gen5065, jeremy: Good for maintenace
>          pref:0.10  max:0.10  rev:1.00  conf:1
>          pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
> problem: conflict%gen5055, roger, NOT-REQUIRED  HARD
>       Heating coil not required
>          pref:0.10  max:0.25  rev:0.40  conf:1.00
>          pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> Proposal was NOT accepted
```

Selection:

```
>>> storageSelProp%gen4769, jeremy:  atmosTank%gen4770,
> evaluation: gen5052, roger: Not as expensive as pressure vessel
>          pref:0.25  max:0.25  rev:1.00  conf:1
>          pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen5062, jeremy, NOT_POSSIBLE  HARD
>       Cannot store hot toluene in atmos. tank
>          pref:0.35  max:0.40  rev:0.88  conf:0.30
>          pv:nil nil 0.88 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> storageSelProp%gen4776, jeremy:  floatingRoofTank%gen4777,
> evaluation: gen5053, roger: Not as expensive as pressure vessel
>          pref:0.25  max:0.25  rev:1.00  conf:1
>          pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen5063, jeremy, NOT_POSSIBLE  HARD
>       Highly volatile and toxic fluid, floating roof not appropriate
>          pref:0.00  max:0.00  rev:0.00  conf:1.00
>          pv:nil nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> storageSelProp%gen4783, jeremy:  pressureVessel%gen4784,
> evaluation: gen5054, roger: Pressure vessel expensive
>          pref:0.10  max:0.25  rev:0.40  conf:1
>          pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen5064, jeremy: Costly system to get tank to discharge to pressure vessel
>          pref:0.00  max:0.20  rev:0.00  conf:1
>          pv:0.00 nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10
> accepted storageSelProp%gen4769 <
```

Discussion:

The second tank in the storage set was also accepted to be an atmospheric tank. No conflict exists with this design case.

ATMOSPHERIC_TANK atmosTank%gen4770

No further design required.

PIPE gen4019

No further design required.

TRANSFER transfer%gen71

No further design required.

TRANSFER recycle%gen77

Attributes:
..fluid-contaminants particulates; unit-process-downstream MAIN::storage%gen65; unit-process-upstream MAIN::still

Parametric design:

>>> transParamProp%gen113, jeremy: transfer%gen114,
> accepted transParamProp%gen113 <

Discussion:

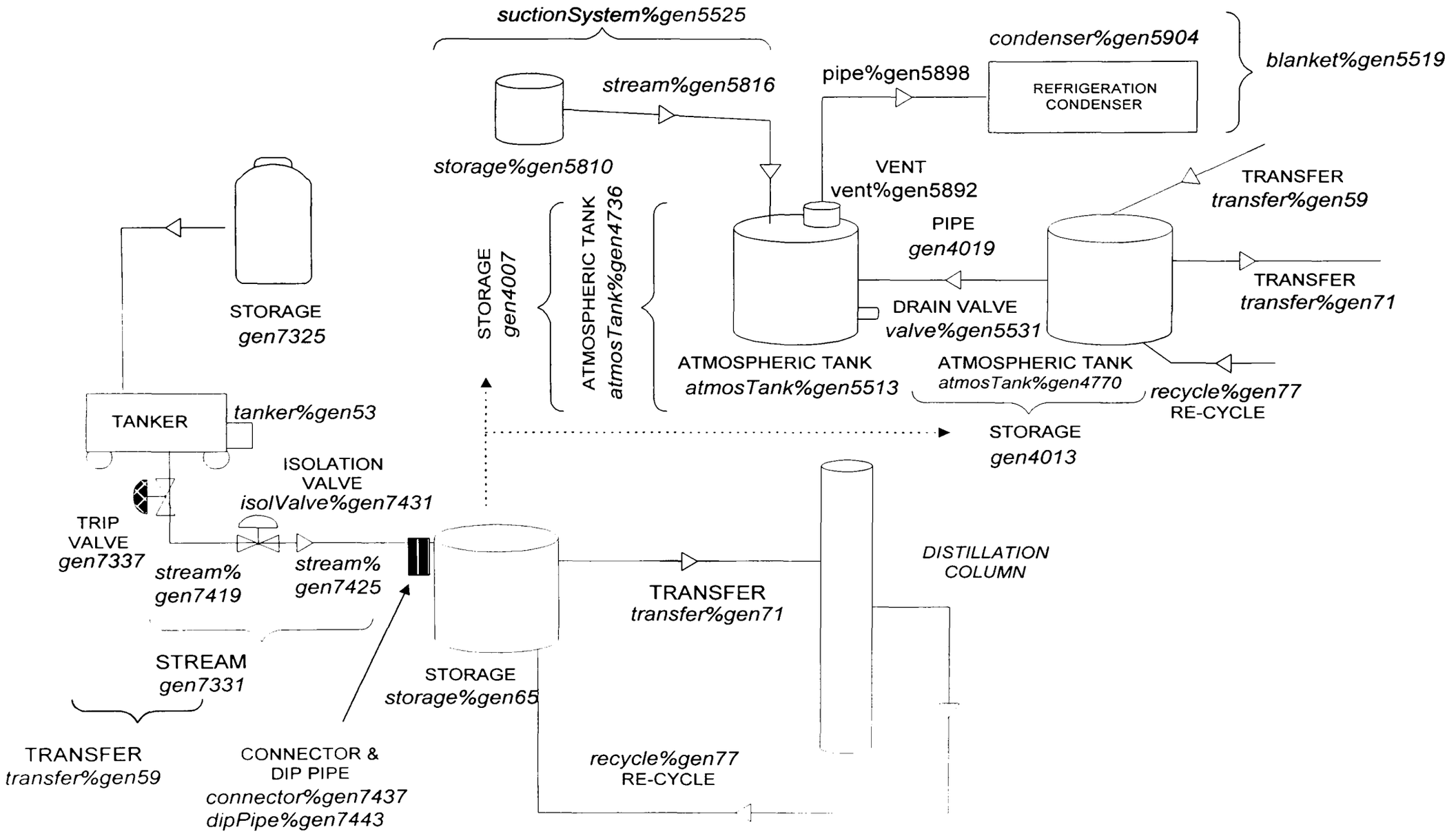
The transfer mechanism connected the distillation column to the toluene storage tanks. Particulates were expected in this re-cycle which have been identified with this proposal.

TRANSFER transfer%gen114

No further design required.

7.4.3 Completed Design

The following diagram depicts the complete design case where each of the design sections covered above have been superimposed into one another. The appropriate design object reference numbers have been identified on the diagram to enable cross reference to the design hierarchy found in appendix I.


suctionSystem%gen5525
stream%gen5816
storage%gen5810
pipe%gen5898
condenser%gen5904
REFRIGERATION CONDENSER
blanket%gen5519
VENT
vent%gen5892
TRANSFER
transfer%gen59
PIPE
gen4019
TRANSFER
transfer%gen71
STORAGE
gen4007
ATMOSPHERIC TANK
atmosTank%gen4736
DRAIN VALVE
valve%gen5531
ATMOSPHERIC TANK
atmosTank%gen5513
ATMOSPHERIC TANK
atmosTank%gen4770
recycle%gen77
RE-CYCLE
STORAGE
gen4013
STORAGE
gen7325
TANKER
tanker%gen53
ISOLATION VALVE
isolValve%gen7431
TRIP VALVE
gen7337
stream% gen7419
stream% gen7425
STREAM
gen7331
TRANSFER
transfer%gen59
CONNECTOR & DIP PIPE
connector%gen7437
dipPipe%gen7443
STORAGE
storage%gen65
TRANSFER
transfer%gen71
DISTILLATION COLUMN
recycle%gen77
RE-CYCLE

The refrigeration condenser was presented as a proposal in the last few minutes of the design meeting as a possible approach to resolving the problems identified with the thermal oxidiser and the flare system. The refrigeration condenser was therefore not discussed in any detail in order to identify any problems with the proposal. CDEX has therefore not accounted for any problems with the refrigeration condenser and therefore it has not been penalised by CDEX to an extent where the proposal has been dismissed, but generally ranks higher than the other proposals. Both Roger and Jeremy were enthusiastic about the proposal when it was considered, even though it was not covered in any detail. The difference between the generated design and the main points of discussion regarding the thermal oxidiser therefore is not considered to be important in reflecting the capabilities of CDEX to resolve conflicts to rationale conclusions.

In the design meeting the engineers did not agree on the exact configuration of the storage mechanism, whether recycle should go to one or both tanks. Good arguments were presented for both cases. A recycle to both tanks would be very expensive as some form of flow control would be required, whereas if flow was to recycle to a single tank the flow will not split (a buildup of re-cycle particulates in a single tank) which could cause problems. The problems were not elaborated in the design meeting and therefore not covered by CDEX. CDEX chose the recycle to the single tank, which may or may not have been the solution accepted by the engineers given more time to analyse the problem. It was however mentioned during the design meeting that with a flow control mechanism, a good mix cannot be guaranteed, in which case a return to a single tank may not be the most logical option as expense would then be the main discriminating factor. The choice in this case can therefore be considered a rational choice given the information available.

7.4.4 Quickest design using case scenario knowledge - Design case 2

Appendix J gives the CDEX design for the same requirement and knowledge bases as used in the first design case. The exception in this case was that time was identified as the most important parameter, and a design was requested. The detailed design generated by CDEX for this case scenario is described in appendix J.

This section will cover the differences between the two detailed designs and cover the rationale behind why the design is different.

System for off loading from tanker

In both the first and second test cases, the same system was selected to blow through the tanker with a trip valve to prevent pressure blow through.

A conflict was identified in both cases, although the approach to conflict resolution was different.

In the first case, where quality of solution was an issue, the following is an ordered list of conflict resolution strategies applied to resolve conflict:

> compromise, consensus, compromise, consensus, compromise, consensus, compromise, consensus, compromise, consensus, compromise, consensus, majorityRule, majorityRule, majorityRule, majorityRule, compromise, consensus, majorityRule, majorityRule, majorityRule, compromise, consensus, majorityRule.

In the second case, where time was an issue, only the following single strategy was applied:

> compromise

It was expected that the second case would not have spent as much time in conflict resolution as the first case, as the quickest solution as opposed to the highest quality solution was required. The difference however was bigger than expected. A small subset of the trace file[5] that was generated during the design process in the first case identifies the issue involved.

```
REFINEMENT:  new proposal [MAIN::transFromTank%gen138]
REFINEMENT:  new proposal [MAIN::transFromTank%gen157]
REFINEMENT:  new proposal [MAIN::transFromTank%gen190]
REFINEMENT:  initial proposal request complete
REFINEMENT:  received proposal ready to review
REFINEMENT:  received proposal ready to review
REFINEMENT:  received proposal ready to review
REFINEMENT:  new proposal [MAIN::gen292]
REFINEMENT:  received proposal ready to review
REFINEMENT:  refinement chose proposal [MAIN::transFromTank%gen138]
*negotiation: Permission to proceed denied synthesisedRefinement%gen63
*...
*negotiation: Permission to proceed granted synthesisedRefinement%gen63
REFINEMENT:  new proposal [MAIN::gen660]
REFINEMENT:  received proposal ready to review
REFINEMENT:  refinement chose proposal [MAIN::gen660]
.....
.....
.....
REFINEMENT:  refinement chose proposal [MAIN::gen292]
```

The lines marked with a '*' are of interest in this case. These actions are repeated many times during the design phase. Permission to proceed is denied when the solution is not improving as expected, and the more detailed design process therefore requests permission

---

[5] The full trace file has not be included as an appendix due to its size - over 30 thousand lines. The sample shown has had references to proposals, refinements etc removed to improve legibility.

to proceed. Each time 'permission-to-proceed' is requested, it is denied as the design route is not as good as expected. However, after reviewing the other alternatives and re-evaluating the conflict resolution strategies, the same design route again proved to be the most promising. This is reason behind the trace file having many different conflict resolution strategies depicted for the refinement. In the second case example, permission to proceed is granted because the weighting of the proposal on the design path is greatly improved due to the amount of time spent on the design path. Permission to proceed is therefore granted in all cases, and therefore only one conflict resolution strategy has been applied.

In this comparison, it is noted that the time required for the strategy to execute did not effect selection of the conflict resolution strategy. A compromise strategy was selected in both cases and conflict was resolved. The problems occurred when problems were found late in the design phase, and the time spent on the design path became a more critical factor in the choice of a design path in the second case therefore avoiding conflict.

Synthesis of the storage requirement to two tanks

The solution derived from both cases 1 and 2 was the same, two tanks connected by a pipe. The approach in which conflict was resolved however was different, although both strategies had different approaches to resolving conflict.

The second case, the time critical example applied the following strategies:
compromise, consensus, majorityRule, compromise, consensus, smoothing, majorityRule, compromise, consensus, majorityRule, smoothing, compromise, consensus, majorityRule, smoothing, compromise, consensus, majorityRule, smoothing

The first design case applied the following strategies:
smoothing, smoothing, compromise, smoothing, consensus, compromise, consensus, compromise, consensus, compromise, consensus, smoothing, compromise, consensus, smoothing, majorityRule, majorityRule, compromise, consensus, majorityRule, majorityRule

From a review of the design trace in the first case, 39 different cases where identified where permission was requested to proceed. Permission is only requested to proceed when design is not as good as expected accounting for the time spent on the design path. From looking at the generated design (appendix J) it can be seen that the more detailed design stages have a generally lower design value (compare the evaluations in the design of the blanket, compared to the evaluations for the synthesis of the storage requirement (two tanks connected with a pipe). The evaluations for the synthesis of the storage synthesis are as high as 1.0 (perfect) as compared to 0.7 which was Roger's view of the best blanket. There was therefore considerable conflict and design iteration with regard to selecting the design path and both the first and the second design case had problems here.

A point to note however with the time constrained case (case 2), is that the initial strategies applied to resolve conflict were the ones that are expected to resolve conflict in a shorter time frame: compromise, consensus, majorityRule, compromise, consensus etc. The strategies applied where time was not critical (the first case) were rather different: smoothing, smoothing, compromise, smoothing, consensus etc. This shows that the quality case wanted to justify a design approach by bringing more information into the design approach (smoothing) as opposed to selecting the quickest approaches like consensus and majority rule.

Difference in selection of blanket system

In the first design case, the refrigeration condenser proposal was the adopted approach. This approach was also favoured in the design discussion as a good option. The second case however selected a mechanism to discharge by flare, which when considered in context of the design case, was not favoured because potentially - in the future - toluene will not be allowed to vent to atmosphere.

In the time critical approach, a single compromise was made. In the quality approach, a compromise was made, then a consensus.

The quality approach attempted to compromise on the proposal with the condenser (which was accepted), whereas the time driven approach attempted to compromise on the proposal with a fan blowing through to the oxidiser. This difference is attributable to the confidence having a high rating in the evaluation of a proposal, and preference has a low rating. The fan and oxidiser proposal, although not the most liked proposal, has a high confidence compared to the other proposals (.6). This high confidence, compared with a medium preference picked the solution out for analysis. The compromise approach was applied. Through accepting this compromise approach, the agent is accepting a particular loss in its ideals and this acceptable loss in ideals is recorded. When the strategy has been applied, it was found that another solution was now acceptable, accounting for the loss in the agents ideals, therefore brining in other options. The proposal accepted was therefore not the proposal on which the compromise was made. The flare was not a bad proposal although it was thought that potentially in the future it would not be possible to flare toluene due to the poorer combustion provided by a flare.

7.4.5 Observations for both design cases

Issues regarding conflict

The negotiation mechanism operates on the basis of the individual engineering values, and trades these values off against each other in the hope of determining a solution most

acceptable from all perspectives, even if small sacrifices have to be made. There are certain issues regarding conflict however which the mechanism does not address:

- an agent cannot suggest that another agent is wrong with regard to the knowledge applied. e.g. a process engineering agent could not tell a mechanical engineering agent that his basic knowledge of heat transfer in a pump is incorrect.

This problem is due to the fact that an agent cannot see another agent's knowledge. One has purposefully not enforced any standard, thereby enabling different technologies to be utilised. If two agents have knowledge on the same subject, which is what is implied by one agent 'knowing' that another was incorrect, then the knowledge could be applied in both cases to the design problem and if the results are different a conflict would arise. Although the conflict may be unnecessary, if there is a fundamental flaw in the other agent's reasoning to enable a bad solution to be selected, the correct agent could raise a hard conflict thereby not enabling the bad solution to be accepted. In human terms, when one has this problem, one may refer to books, or the engineer with the most experience or authority may decide. Although it is possible to factor in other rules to the negotiation process, such as the number of rules defined for the agent (i.e. a potential measure of experience!), the problem is somewhat complicated and presents an interesting problem for future research where there is no common knowledge representation format.

- where an agent overstates the importance of his proposals. It is possible that when an agent puts forward his proposals, he states that it satisfies all the goals to the highest degree. In this case the negotiation mechanism would obviously lean towards accepting the proposals from the agent that overstates his case. Obviously proposals where the other agent has stated hard conflicts, the proposal will not be accepted, but the agent who faces the another agent that overstates his case will loose out. Similar problems exist in the human workgroup as covered in the chapter on human forms of negotiation. A person for example may overstate their case for say a pay rise, so that if a compromise is made (a middle value) they are likely to be better off than if they asked for what they wanted (in which case they would likely get less than they wanted if a compromise is made). In computational terms this is a difficult problem to resolve. One could go part way towards resolving the issue by enabling an agent to state his values (objective hierarchy) at the start of the design process and let the agent specify which goals are considered in each rule. This would therefore prevent the agent from specifying a preference value for a proposal greater than that of the total preference of the objectives identified to be considered in the rule. One could identify unreasonable agents by those agents that specify all their objectives to be considered in all rules! This however does not do much to resolve the problem, as agents can still prefer a proposal to the maximum extent considering the objectives they have specified they will apply. Another solution would be to have some fuzzy damping factor, say 'exaggeration', that takes the mean of the preference of the proposals put foreword, and utilises this factor in

modifying the preference itself. So for example if an agent always overstates his case, his mean preference will be high, resulting in a high exaggeration factor, which can therefore be used to reduce all his preferences considered for by the negotiation mechanism. This is not a very scientific approach, and is likely to be affected by agent experience and other factors. Further work would be required on the development of the mechanism to cope with this problem.

From a review of the issues above, it can be seen that one has suffered similar problems to those identified in the human forms of negotiation. It is interesting to find that the experience and knowledge gained from negotiation in the human workgroup has been utilised, yet have suffered the same problems in its application when developed in computational form. The research has not therefore developed a computational negotiation mechanism that can be improved to surpass that of a human workgroup, but has accomplished the objectives of enabling computational agents to apply the same negotiation reasoning, thereby improving the speed of the process and enabling disparate knowledge based systems to produce collaborative designs.

High level of iteration in the design

Some stages of the design saw a great deal more conflict than was expected (e.g. design case 1, transfer%gen59). In these cases conflict was quickly resolved, although design progressively got poorer, and eventually the design path was not allowed to be explored further. When the negotiation mechanism reviewed all the available options, it was found in some cases that the path where permission to proceed was refused, was still the best alternative open for analysis and therefore design was requested to continue. This re-evaluation of a design path lead to further analysis of conflict which is the reason why some design objects were difficult to get accepted. These problems basically stem from a poor evaluation of how design was expected to turn out by the agents in the early design phase.

It is expected that if agents are built around the basic premise of design functions evaluating criteria for a particular design objective, the number of discrepancies found will not be as high. Through using judgement on how engineers weight particular solutions based on the design meeting discussion, any discrepancies not in favour of the design, and larger than the factor accounting for the time, will be the cause of iteration in the design. If functions, rather than judgement were applied in determining the quality of design, the evaluation of design proposals will be a more systematic process, therefore the more consistent approach to reviewing alternatives will not be the cause of unwanted iteration.

Unexpected proposals

Using the framework it was possible to identify knowledge that was not explicit enough. For example, when reviewing a storage mechanism, a heating coil was always proposed for consideration. This consideration was applied to all storage mechanisms regardless of

type. From analysis of the design meeting discussion, this does not seem unreasonable. However, CDEX put forward this proposal for a nitrogen supply (which must be somewhere stored in a tank). In this case the knowledge could be considered to not be explicit enough, or it could be considered that heating coils may be appropriate for a nitrogen supply. The design meeting did not elaborate on this issue, although CDEX picked up the issue because the agents represented their interests in the design of a storage mechanism regardless of contents. It is considered that there is some cross here between the more generic knowledge which showed itself in this case scenario, and reasoning by analogy. This however is not a topic for further analysis in this research.

Few strategies applied

In the design case, very few strategies were applied. Mainly compromise, consensus, majority rule and smoothing were utilised in resolving conflict. This is due to the fact that most of the conflicts identified in the design case were single issue, in which case several strategies were instantly not applicable. For the approach to utilise more object values, it is expected that an agent should maintain a single hierarchy of objective values and keep this up to data as design progresses and more information is brought into the design path. This would enable the agent to comment on other values more easily than at present (i.e. having to review all goals for a single proposal).

The CDEX design case was formulated from the design discussion and most of the issues addressed as potential problems in a proposal were single issue. These issues were generally the major issues which overshadowed other considerations in the meetings. The absence of these 'other' considerations therefore reduced the number of objectives one could identify as important when defining the rules. The solution accepted as a result of the design process was rational, and therefore it is not considered important that not all strategies were applied.

Two agents

Only two agents were involved in the design case. The mechanism adopted is applicable for any number of agents as the functions for resolving conflict were developed for multiple agents in mind. The tests covered however did not tackle multiple agent issues, primarily due to the difficulties in putting together and collating the expertise from such a large team. Scale up problems may be an issue although this has not been identified in any of the tests performed.

7.5. Summary

A test scenario of a real design case was run through the CDEX framework after encoding the knowledge and beliefs of the two agents in the design case. The results were quite pleasing in some respects as the designs were very close to the design explored during the design discussion. Some small differences existed as noted in the result analysis and justifications were provided as to why the framework reasoned about the problem the way it did. Where the framework did select a different proposal from the one discussed in the design meeting, the choice was a difficult choice anyway. It is important to note that the more difficulty the agents have in making a choice, the less the choice really matters (as they are close to being equal).

The framework consistently applied similar strategies: compromise, consensus, and smoothing. These strategies were popular due to the fact that the agents' knowledge was represented mainly with regard to a single issue rather than what the design is like from the agent's entire perspective. It is expected that this will normally be the case when codifying knowledge from a design meeting as a great deal of things are left unsaid. The points made in the meeting are usually made with regard to the biggest mitigating factors, for example, Jeremy may be concerned with an atmospheric tank storing hot toluene (the safety factor) rather than commenting that the atmospheric tank is also cheaper than other types of tank.

Considering the quality of knowledge gathered during the scenario discussion, the system utilised the knowledge and came up with a good result. It is difficult to assess how an example with more agents involved will work. The strategies can deal with the views from many different agents. The issue is with regard to there being more conflict which is more likely when more viewpoints are involved. This greater level of conflict will affect the time spent in attempting to find an appropriate solution. In this case the system is more likely to adopt the quicker conflict resolution strategies (e.g. compromise, majority rule) in fewer iterations than that normally applied with only two agents. A way around this discrepancy would be to allow the number of agents to be accounted for when accounting for the time factor associated with particular strategies. The issues here can only be explored through the development of a system to encapsulate many different viewpoints. This analysis is likely to be very costly in terms of effort.

Chapter 8. Conclusion

8.1. Review of the 'CDEX' approach

Expert systems have provided the mechanism to provide expertise to engineers that may be outside their personnal scope of knowledge, thereby improving the quality of their efforts. This ability to store and recall knowledge in a context specific to the engineer's task at hand is undoubtedly of extreme value. Not only does one have a mechanism for collecting the best expertise in one place, but also the ability to 'copy' and provide this information to as many engineers that require the support at little cost. The knowledge based system is acting as a tireless expert available to all who need it.

The development of these systems has progressed as individual entities, or 'collections' of expertise. These electronic expert entities tend to provide support in specialist tasks. Design tasks however encapsulate a great deal more expertise than that covered by any of the individual entities built so far. For a system to have a handle on the bigger picture, it must have complete knowledge of all disciplines and be able to understand and resolve the inherent inconsistencies and tradeoffs between all the information available. One may consider the development of a knowledge based system with such an all encompassing outlook to be an impossible task. Elaborating all the tradeoffs that have to be made would be extremely difficult, not just because of the magnitude of the problem (the number of potential combinations to consider would be enormous), but getting the experts to agree on the tradeoffs would probably take a horrendous amount of time to resolve!

Do we need systems with such an all encompassing outlook? From a design engineering point of view one can see its value. The design decisions made early in the design phase are typically larger grained, that is, the decisions have a far greater impact on the final complete plant than the decisions made later in the design. The ability for an engineer to access the impact of his design decisions on the rest of the design would enable better (more informed) decisions to be made and therefore have a more optimal end result. From another perspective, a system that could integrate the knowledge of a whole variety of domains could automate the design process and therefore produce a complete design from an initial requirement. This however would require a considerable amount of knowledge and would be very expensive to develop.

How are we to progress towards the development of a system with such an all encompassing outlook? The problems as pointed out of building a single large system appear insurmountable. The answer, it is believed, lies in integrating disparate entities, with different expertise, and guiding them to work on the same design problem. It is known that the development of systems in isolation, by individual disciplines, has had some successes and therefore the successful principles can be applied in the development of the many different design experts required. The integration of these viewpoints however

will cause conflict. Depending on the viewpoint and values of the engineering discipline building an entity, these values will present themselves in the behaviour of the individual expert entity and therefore create conflict and differences of opinion which will have to be resolved. The development of a system to enable these differing entities to cooperate must therefore be able to communicate requirements in a manner comprehensible by all, and also mediate over the resolution of conflicts, the 'negotiation' metaphor. The principle behind the success of an environment to integrate disparate entities and resolve 'conflict' is the premise that the smaller entities can be built over time with incremental benefit being attained, with additional benefits of access to larger areas of expertise, potentially multiplying the benefits of the system.

Through resolving the problems of negotiation and enabling differing expert entities to cooperate one is providing the important next step in the development of computational systems to provide expert advice. A framework will enable the differing sets of expertise as well as disparate technologies (mathematical & logical) to be brought into the resolution of design problems. This research covers the development of a mechanism for enabling cooperation through the sharing of engineering 'values'. These values encompass the feelings that an entity has on aspects of the design - the 'objectives'. Differing entities may have differing and shared objectives, and support these varying objectives to differing degrees. A safety engineer for example will probably consider the decisions regarding 'safe operation' of the plant over that of cost, whereas another engineer in a different discipline may have the reverse perspective. This research adopts the ideas behind utility theory, the mechanism through which the importance of objectives is measured by numerical weights, and enables disparate entities to cooperate through sharing of these values. A research into the human theories of negotiation was performed (compromise, drop goals of low importance etc) and computational forms of these human theories developed around the basis of manipulation of these utility weights. The justification for codifying these human theories is our belief that a mechanism that follows the human line of reasoning and justification is more likely to lead to results that can be accepted by the human engineers using the system.

The results produced by CDEX were very close to the design explored during the design discussion. Differences in approach were noted between the design meeting and the results from CDEX, but these differences were unclear as several design routes were explored by the engineers, and the actual design path to be taken in the design was not agreed. In order for an approach to be finally agreed, considerable man hours would have to be spent looking up knowledge to support a particular design approach. This knowledge was not available in the design meeting. It could be considered that the approach adopted by CDEX was 'rational', due to the fact that engineers would not discuss an option in detail if the approach was not at all acceptable, together with the fact that the solutions proposed by CDEX were not explicitly discounted in the design discussion.

The example design case did not utilise a great number of the conflict resolution strategies, mainly due to the fact that the knowledge identified during the design meeting was

normally on a single issue. Many of the resolution strategies were appropriate where the conflict had multiple issues (abandon less important goals, goal integration etc). The aim of the research was not to utilise the many different strategies, but to derive an acceptable rational solution to design problems. It has been shown through the case scenario that this can be achieved by utilising just a few strategies: majority rule, compromise, consensus, and smoothing.

8.2. CDEX Approach in another problem domains

CDEX is a computational approach to support the design process and has shown good results for the particular scenario developed in this research for the chemical process industry. A question remains however as to the applicability of the approach in other industries.

The knowledge defined within the agents developed within the scenario is specific to the design problem and therefore is only applicable for the types of problem which address the problem of storing toluene. The CDEX approach however is based on a specific design approach, using object oriented principles, applying a simple grammar, with a general negotiation approach.

The approach to negotiation is a general one, and was developed outside of the problem of designing process plant. The basic approach of using an objective hierarchy to identify values was a prescriptive approach borrowed from the field of decision analysis. Decision analyses develops a general theory of decision making that can be applied in any domain where decisions have to be made. The approach to resolving conflict by trading off agent values was developed from an analyses of human conflict resolution techniques. These techniques are also general techniques available in the field of negotiation and how conflict is resolved. It is believed therefore that the approach to negotiation taken in this research can be transferred successfully to another field.

The object oriented approach to representing design objects is also quite general. Many applications are being developed based on object oriented principles as a general approach to problem solving. A constraint identified with the approach however is the granularity in which objects are expressed. The approach CDEX has taken on design is that it can be broken down into three basic processes: design selection; parametric design; and design synthesis. These design processes work on whatever design 'objects' are available in the design environment. The approach does not permit new design objects to be created. In design terms, this is what could be referred to as innovative design, although the term innovative is used in a loose sense as it is possible to refer to a design as innovative even if was built from pre-defined components. This inability to produce new objects is unlikely to be useful in an environment where brand new products and ideas are required.

The CDEX approach is therefore more likely to be viable in domains where components are readily identifiable, and there is good pre-defined knowledge and experience in using and building solutions based upon those components. This not only covers process plant design but many different types of industry that has a civil engineering function. Before attempting to apply CDEX in another industry, an assessment should be made of the knowledge available, whether the domain includes a large element of creative design, and whether domain components are readily identifiable. The CDEX approach can be used in conjunction with a creative design element but obviously knowledge will not be available to assess the new component from multiple design perspectives.

8.3. Potential Uses for CDEX

Aside from a concurrent engineering tool to provide design support from the viewpoint of other disciplines, CDEX - because of its ability to derive a design - could be envisaged to help in the following areas: the application of standards, safety analysis, the 'design warehouse' concept and plant costing.

Application of standards

The application of standards is important in the design of plant, to avoid the potentially costly mistakes, whims and fancies of the engineers involved throughout the design process. Standards however are complex documents, voluminous, and difficult to maintain. Training the design engineers in application of the standards, and keeping them up to date, is also difficult. Knowledge based systems can facilitate and review the design process at each stage as it proceeds, thereby validating and ensuring compliance with the standards. Although there are a number of complexities involved in developing systems to perform this task, as well as requiring considerable effort in development, knowledge based systems have the capability to support this task.

Our current society has placed an important emphasis on the enforcement of standards. Liability for failures has lead to the increasing number of standards and guidelines that are copious and complex to handle. An excellent example of this apparently occurred in Saudi, whereby the standards were supplied and locked in a guarded room so that they could not be misplaced. Engineers were expected to review the information in the room as and when necessary. As you can expect, without ease of access and the volume of data available, the archive was little used.

It is common for equipment vendors to receive a hundred page document covering how to specify and select a product. With little time available to quote, engineers are not going to examine details, if they even have time to open the standards at all. Contracting companies are often requested to adhere to special company standards but due to time constraints they either apply their own standards or internationally agreed standards throughout the design - the company standards 'go in the cupboard'. These types of cases

are all too common where one is consistently striving to avoid liability, and at the same time complete jobs faster and more efficiently. Without a technology to support the engineers through this mass of knowledge, the dilemma will continue. Vast amounts of effort will be wasted and capital spent in the production of documentation which neither aids the engineers, nor utilizes the massive pool of invaluable engineering expertise which has been learned.

Safety analysis

Accessing the potential risk and hazards continuously will reduce re-work and compromise. Currently reviews are staged throughout the design life cycle (e.g. early process flowsheet, preliminary piping and instrument diagram). Problems identified at these staged reviews can cause design re-work back to the stage of the previous safety study if safety issues were not considered. A continuous review process is therefore required in order to avoid this iteration. Currently, a design engineer will account for the many different design considerations based upon his experience. A safety review team (or HAZOP team) will bring experience from several disciplines, and therefore bring to bear a far greater level of experience into the review process. Ideally this 'knowledge' should be available when the design decisions are being made - although the costs in engineering effort would be extremely high. An automated review process would highlight the safety issues and concerns when any decisions are made, and therefore reduce the potential for error at a safety review stage.

Such a safety system would be built up over time, and be especially useful for engineers new into the profession where their knowledge of the life cycle concerns is not as vast.

Design 'warehouse'

Although there are many chemical plants producing the same products, the design and construction of the plant is typically unique. These unique designs are required to account for the differing availability and quality of feedstocks, by-products produced (to meet a demand, or to meet legislative restrictions), where the plant is built, and the quantity of product required. Although a plant may be unique, the components and systems that make up the plant may be standardised. From reviewing the plant design hierarchy (Figure 8), it can be seen that the majority of components at the lowest level are standard (e.g. pumps, valves, pipes, fittings, connections etc). However, it is common nowadays to find more systems available for purchase as complete packages of components. These systems meet a requirement normally found in the higher levels of the plant hierarchy. Such a system may be a heat transfer system, which can be purchased as a package of valves, pumps, vessels and other items that make up the system. The advantage of purchasing these packages is to avoid design effort, obtain a single cost for the package, and have a pre-safety checked and tested system.

As was mentioned previously, design synthesis is a complex function. The availability of

these pre-defined system designs to meet requirements in the plant hierarchy will aid the automatic generation of plant designs for review. The hierarchy appears to be a good mechanism for organising a structure and classification of pre-defined system designs. Such a database of design cases can be built up over time, and rules defined to differentiate between the different cases to ensure that the correct system is selected when a design synthesis is requested.

Plant costing

Costing of plant can be a difficult activity due to the vast number of 'unknowns' at the early stages of design. As expected, the only true estimate of the cost of plant is obtained when the plant construction is actually complete - after all the detailed design and construction is finished. General rules of thumb, order of magnitude reasoning, and past case histories are good examples to aid in the costing effort. However, if a system could, at an early stage of the design process, automatically produce detailed designs from an analysis of the requirement, more accurate cost estimates would be produced at an earlier design stage. Such a system would need a large repertoire of stored design cases, sizing programs, and selection systems. The resultant design could then be reviewed on-line by a group of engineers. This design would in essence, represent the potential end result. As such the suggested design route could be intricately examined. Decisions as to whether this design would be a "reasonable" solution could be made. A system with this capability would improve the reliability and accuracy of cost estimates.

8.4. Future research

The approach taken in the development of the strategies could be termed 'fuzzy'. The 'rules of thumb' applied in the human domain in selecting a strategy, were translated into numerical form using subjective, although simple weighting criteria. Further research in the field would explore more detailed approaches to determining more accurate weightings for these factors. During the development of the framework, many issues were noted with regard to how the approach could be improved, and these improvements are noted in the following paragraphs.

8.4.1 Conflicts are dealt with between agents not proposals themselves

In CDEX the resolution of conflict is managed between different agents. It is possible however that agents have conflict internally to their own rule bases. Minsky [Minsky 86] portrayed individuals as potentially having internal conflict: "Sometimes we regard ourselves as single self-coherent entities. Other times we feel decentralised or dispersed, as though we were made of many parts with different tendencies.....we all have feelings of disunity, conflicting motives, compulsions, internal tensions and dissensions. We carry on negotiations in our head." This is an interesting view, and could possibly be put down

to us not having had the time to review and understand which potential solution most satisfies our understanding of how one thinks things should be done (our ideals) although it would be difficult to prove that this was the case. Ideals do of course change. Discussions [SfK 92d] have identified that sometimes there is a 'flavour of the month', where particular attention will be paid to designing things differently depending on external factors (latest news events on plant failures). Changes to engineering designs may be made to mitigate these problems even if the risk is low and does not justify the expense. This new approach to design may change back to a more rational approach when other factors become more important (e.g. cost). CDEX operates on the basis that a single objective hierarchy exists for one agent, i.e. each individual has a single consistent set of values. This approach however can be modified to simulate the above changes in design approach by constantly changing the value weights in the objective hierarchy.

8.4.2 Extending the types of design process

The framework enables an object to be synthesised, parametised, or selected. This mechanism however has been found to be too rigid. A further type of design should be considered, that is 'additional requirements' which in the development of CDEX was considered under the synthesised proposals. For example, if a tank is to have for example, a limit control switch, this would be recorded in the current CDEX system as the synthesis of a TANK object. It could however also be considered as a synthesis of STORAGE, and therefore CDEX has had to lay down rules regarding where devices are synthesised. At first this does not appear to be a problem, and synthesis provides the mechanism for doing this. However, the system is a little cumbersome as anyone who synthesises the TANK object, must also include the possibility of a tank control system, and include this in the proposal, even though he may not know about tank control systems. Initially, a 'dictionary' type approach to resolving this problem was considered. When a tank is created, the system could automatically generate the possibilities for a tank control system and other extensions to the tank. These proposals would then be reviewed by other interested agents, and either refined or not. The dictionary would act as the book of general issues which have to be considered when specific design objects are placed in the design environment (an additional requirement would be maintenance procedures for example). This approach would work, although now another maintenance consideration has been introduced into the system, which may have to be agreed globally and which should be avoided. Another possibility, and more clean approach, is to have another category of design apart from synthesis, selection and parametric as mentioned above, the 'additional requirements'. In the case of the TANK, the tank would be proposed, and an agent would propose a limit switch or control mechanism as an additional requirement to the TANK. This additional requirement can be considered as a separate design problem completely, and proposed as a design requirement without reference to the tank. In this case the system would design the control mechanism independently of the main plant design. As long as the first assessment of the problem by the agent retrieving the TANK design parameters was ok, then design of the limit switch and control mechanics would be considered in isolation. There is however a problem with this approach in that there is no

tie back to the TANK from which the basis for the design originated. This means that if the TANK is removed from consideration, CDEX would not know to remove the control mechanics from consideration as well, as it considers it to be a completely separate design. Having the 'additional requirements' of an object would enable one to link any additional designs back to the originating requirement (i.e. the TANK). and therefore modifications in the spec of the tank would be reflected back to the additional design requirement.

Having the additional requirements also provides an additional benefit. At the moment. the leaf nodes in the design structure are identified as the physical entities. Work is based on the premise that a synthesis, parametric design, or selection is a pure replacement of the parent object. This is a logical step. However one finds oneself making up object names in the case for synthesis proposals which is not an entirely clean method of design. For example, if one decides on having a control system for a tank, and then synthesises the tank, the synthesis proposal must include the control equipment together with the tank as one is 'replacing' the parent synthesis proposal. It must also be ensured that agents do not then redesign the tank without considering its parents and checking that the control mechanism already exists. Specifying the control mechanism as an additional requirement, makes the solution more clean, as essentially one does not wish to replace the tank by presenting another synthesis proposal.

Changes that would be required to the code would be to ensure that the additional requirements were included in the evaluation of the design path (together with the synthesis, selection, and parametric design). It is important that each design path can be independently accessed as the design progresses. This is different from the normal hierarchy of components (the is-a hierarchy), as a design path cannot be accessed from this as things like control systems in the tank hierarchy for example are not included.

If one considers the synthesis and selection as replacements, then proposals can be negotiated together and not as different entities. Only one of the proposals can be selected if one considers them as replacements as in the feature of 'additional requirements'.

8.4.3 Extending the repertoire of agent interests

It was found during population of the agent knowledge base that the number of events an agent could respond to was quite restrictive. Currently the agent represents his interests and he is informed when the particular interest is appropriate with the current design state. These states occur when either an object is accepted (as part of a proposal) in which case further design is requested, or when a proposal is put forward for review in which case the agent may be requested to evaluate the design. There are cases however when an agent is specialised on a particular object, say for example vents, and he cannot represent his views on the vent immediately as there is no event to signify that the vent object has been created. Currently if the agent wishes to present views on the vent in particular, the agent has to be either prepared to review all proposals with the potential of the vent being present, or the agent expresses his interest in the design of a vent so that when design is

requested he can put his position forward. This however is not really acceptable because design requests are only put forward after the proposal is accepted. Presenting views after proposal acceptance in this manner causes an unnecessary amount of re-work and the proposal has to be re-reviewed.

This problem is not as straight forward to resolve as it may initially appear. The same agent may present more than one view on the proposal (one on the complete proposal, and one only on a part of it) which will lead to issues in integrating the views of the same agent before proposal evaluation begins.

8.4.4 Identifying a root cause of conflict

The 'keyword' approach to describing the cause of a conflict needs to be improved in order to ascertain the effects of more than one hard conflict associated with a particular proposal. Currently if two agents believe the design path will result in problems that prevent design from continuing, one cannot be sure that the root cause of the problem they have both identified through conflict proposals is the same. When one reviews a proposal to determine if it is likely to be successful, the results of this assessment may be very different if it is considered that all the conflicts are different, as opposed to all the conflicts stem from the same root problem.

This problem primarily stems from the fact that agents cannot share knowledge and therefore one cannot determine whether the root cause of the problem is the same. This however may not be the only problem, an agent may have made assessments from the past that particular configurations of design for example just cause problems, without having the exact causal knowledge of why. One can however tackle the former problem by enabling the agent to specify the cause-effect chain for the associated conflict rather than just the simple keyword as defined at present. For example, NO-FLOW for a pipe could be put down to "POWER SUPPLY(fail) -> PUMP (stop) -> PIPE (no flow)" (where x -> y means x is the cause of y, and the bracketed terms denote the deviation in the equipment item). This would provide the negotiation mechanism with additional information which it could use to determine if the conflict being proposed by an agent is mutually independent of any other conflicts identified with the proposal. Issues here regard whether the potential terms for failure (e.g. NO FLOW - the standard HAZOP terms) encompass the complete requirements of the root causes for failure that the agents may consider, although one may be able to disregard the root causes of the cause-effect chains of a conflict share common failures. It is felt that there are many issues here with regard to this problem that deserve more detailed consideration.

8.4.5 Analysis of plant structure

Analysing the system structure is not straightforward and sometimes requires a complex set of statements to determine something that may be considered a simple question to ask. For example, it would not be necessary to develop another blanket system for a tank, if a

blanket system existed already on the other tank containing the same product adjacent to the tank being developed. In order to determine this however , a rule is required to determine if the tank is part of a set, and analyse each tank in the set to determine if any of the tanks had a blanket. A grammar to enable the plant structure to be analysed may be quite a complex grammar, but would simplify the effort required in developing the agents.

8.4.6 Structuring design rationale to ease evaluation by engineer

Assigning the rationale to proposals was rather long winded. For example, if two storage mechanisms were put forward in a proposal, the proposal was good for maintenance as one could be run down for cleaning without shutting down the process, as well as providing problems of splitting the re-cycle to maintain uniformity between tanks. Associating this rationale with just a single proposal is rather long winded. In hindsight it would have been more effective to associate the rationale with the objectives for each proposal. In the example just identified, '"good for maintenance as one could be run down for cleaning without shutting down the process"' could be associated with the 'Maintenance' objective, and the problems of splitting associated with an objective regarding the quality of feedstock. Splitting the rationale in this way also opens the possibility of engineers reviewing a design from a single perspective, for example, an operations engineer would be able to request all the operational issues identified in the design.

8.4.7 Improvements to the approach of representing agent objectives

The negotiation mechanism operates by reviewing and trading off the values that are common among the agents from the high level objectives (e.g. maintainability, cost). Agents however are likely to be applying a more detailed objective hierarchy - although these are likely to be different between agents. Where there are similarities between the different hierarchies however, the negotiation mechanism cannot use these to advantage. For example, if the main point of disagreement between two agents is due to a more detailed objective of cost (e.g. cost of maintenance and initial capital cost) the negotiation mechanism cannot trade off the values of these more detailed objectives as they are not modelled internally in the framework. An improvement to the approach taken in CDEX, would be for agents to present their objective hierarchies to the CDEX framework, and leave it for the framework to assess at what level in the hierarchy trade-offs should be made in any given conflict situation. An argument against this approach would be the degree of effort required in design, implementation, and system overhead, coupled with the issue that the more deep the negotiation mechanism looks to resolve conflict, the more unlikely agents are to share the objectives considered.

8.4.8 Improvements to the resolution strategies

The framework has highlighted the potential to utilise conflict resolution strategies normally applied in the human workgroup, in a computational perspective. The relationship between values and weights of factors determined in the context of conflict,

and the strategy selected to resolve conflict, was based purely on general rules of thumb from analysis of human resolution strategies and knowledge of when the strategies are suitable. Refinements are required to the weights and factors considered, and this analysis is likely to become more important if the knowledge that resides in the system becomes larger and more complex. Small errors in the assessments applied to a specific strategy are likely to become more noticeable if more detail is considered. Techniques are required to derive more exact values for these parameters.

References

[Anson, Jelassi 89]
Robert Anson, Tawfik Jelassi, "A development Framework For computer Supported Conflict Resolution", INSEAD Working Papers, No 89/42, July 1989.

[Ashley 92]
Steven Ashley, "DARPA Initiative in Concurrent Engineering", *Mechanical Engineering Journal*, April 1992.

[Barnard 92]
P.C.Barnard, "The Way Forward", Exxon Chemical Ltd, *World Pumps*, January 1992.

[Bond, Gasser 88]
A.H. Bond, L. Gasser, Editors Overview, *Readings in Distributed Artificial Intelligence*,Morgan Kaufmann, 1988.

[Bowen, Bahler 93]
James Bowen, Dennis Bahler, "Constraint-Based Software for Concurrent Engineering", PROJECT OVERVIEW, *IEEE COMPUTER*, 1993.

[Bowen, Bahler 93b]
James Bowen, Dennis Bahler, "Task Coorindation in concurrent engineering", *IEEE COMPUTER*, 1993.

[Brown, Cutkosky, Tenenbaum 89]
D.R.Brown, M.R.Cutkosky, J.M. Tenenbaum, "Next-Cut: A second Generation Framework for Concurrent Engineering", *Computer-Aided Cooperative Product Development*, D.Sriram, R.Logcher, S. Fukuda, Proceedings of MIT-JSME Workshop MIT, Cambridge USA, Springer-Verlag, November 1989.

[Cleetus 92]
K. J. Cleetus, "Definition of Concurrent Engineering", CERC Technical Report CERC-TR-RN-92-003, 1992.

[Conry, Meyer, Lesser 88]
Susan E. Conry, Robert A. Meyer, Victor R. Lesser, "Multistage Negotiation in Distributed Planning", *Readings in Distributed Artificial Intelligence*, Alan H. Bond, Les Gasser, Morgan Kaufmann, 1988.

[Costanzo 92]
Lucia Costanzo, "Management - Knocking down the walls", *Engineering*, November 1992

[Cutkosky, et.al. 93]
M.R. Cutkosky, R.S. Engelmore, R.E. Fikes, M.R. Genesereth. T.R. Gruber. W.S. Mark. J.M.Tenenbaum, J.C. Weber, "PACT: An experiment in integrating Concurrent Engineering Systems", IEEE Computer, January 1993.

[Cutkosky, Conru, Lee 94]
Mark R. Cutkosky, Andrew B. Conru, Soo-Hong Lee, "An Agent-Based approach to concurrent cable harness design", *AIEDAM* Vol. 8, No. 1, 1994.

[Davis, Smith 88]
Randall Davis, Reid G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving", *Readings in Distributed Artificial Intelligence*, Alan H. Bond, Les Gasser. Morgan Kaufmann, 1988.

[De Bono 89]
Edward De Bono, *De Bono's Thinking Course*, BBC Books, 1989.

[Dormer, McKenna 73]
L.S.Dormer, A.McKenna, "A users view of pump requirements in the process industry". *Process Pumps*, Institution of Mechanical Engineers, 1973.

[Dutcher 91]
D. Dutcher, "Experience of an Integrated Product Development Team at MCAIR", McDonnell Douglas Corporation, St. Lousis, *AIAA - Aircraft Design Systems and Operations Meeting*, Baltimore MD, AIAA-91-3149, September 23rd, 1991.

[Eppinger, et.al. 89]
Steven D. Eppinger, Daniel E. Whitney, Robert P. Smith, David A. Gebala, "Organising the Tasks in Complex Design Projects", *Computer-Aided Cooperative Product Development*, D.Sriram, R.Logcher, S. Fukuda, Proceedings of MIT-JSME Workshop MIT. Cambridge USA, Springer-Verlag, November 1989.

[Field, Forsyth 92]
S. R. Field, J. Forsyth, "Specification of a Software Tool to Improve the Reliability of Pump Installations", *Reliability of Piping Systems*, SFK Technology, 11 September 1992.

[Fischer 87]
Steven M. Fischer, Designing Centrifugal Pump Systems, Feb 16th, 1987

[Goldstein 94]
David Goldstein, "The Distributed AI Toolkit", *AI Expert*, Jan 1994.

[Gopalakrishnan, Pandiarajan 90]
B. Gopalakrishnan, V. Pandiarajan, "Product Design for Manufacturing: The use of

knowledge based systems in concurrent engineering", *IEEE*, 1990.

[Gray, Starke 84]
J.L Gray, F.A Starke, *Organisational Behaviour: Concepts and Applications*, 3rd Ed., Charles E Merrell, 1984.

[Harrington, Soltan, Forskitt 96]
John V. Harrington, Hossein Soltan, Mark Forskitt, "Framework for knowledge based support in a concurrent engineering environment", *Knowledge Based Systems*, Vol 9, 1996.

[Hutton, Ponton, Waters 90]
D. Hutton, J.W.Ponton, A. Waters, "AI Application in Process Design, Operation and Safety", *The Knowledge Engineering Review*, Vol 5:2, pp 69-95, 1990.

[Johns 92]
Bill Johns, "Process Synthesis: Designing an 'Intrinsically Clean' process", *Process Industry Journal*, pp 19-22, November 1992

[Kannapan, Marshek 92]
Srikanth M. Kannapan, Kurt M. Marshek, "Engineering Design Methodologies: A New Perspective", *Intelligent Design and Manufacturing*, Andrew Kusiak, Wiley, New York, 1992.

[Kauders 80]
P. Kauders, "Process Engineering Design - Part 1 & 2", *The Chemical Engineer*, January (Part 1) and July (Part 2) 1980.

[Keeney, Raiffa 76]
Ralph L. Keeney, Howard Raiffa, Decisions with Multiple Objectives: Preferences and Value Tradeoffs, John Wiley & Sons, 1976

[Kimura 89]
Fumihiko Kimura, "Architecture of Intelligent CAD systems", *Intelligent CAD,I - Proceedings of the IFIP Workshop on Intelligent CAD, Boston 1987*, H. Yoshikawa, D. Gossard, North Holland, 1989.

[King, Norman 92]
B.J. King, Dr P.W. Norman. "A Step in the right direction", *Professional Engineering*, November 1992.

[Klein, Lu 89]
Mark Klein, Stephen C.-Y.Lu, "Conflict resolution in cooperative design", *Artificial Intelligence in Engineering*, Vol 4., No. 4, 1989.

[Kletz 85]
Trevor A. Kletz, "Eliminating Potential Process Hazards", *Chemical Engineering*, April 1, 1985.

[Knight, 91]
Barry Knight, "Paperless Transactions in the U.K.", *BYTE*, pp. 59-61. June 1991.

[Knodle 91]
M. Knodle, "Transitioning to a Concurrent Engineering Environment", General Dynamics Space Systems Division, San Diego, CA, *AIAA - Aircraft Design Systems & Operations Meeting*, Baltimore MD. (AIAA-91-3151), Sept. 23rd, 1991.

[Knutton 92]
Peter Knutton, "Design for Manufacture - Winners build teams not empires", *Engineering Computers*, May 92.

[Krishnan, et.al. 90]
V. Krishnan, D. Navinchandra, P.Rane, J.R.Rinderle, "Constraint Reasoning and Planning in Concurrent Design", Carnegie Mellon University, The Robotics Institute, Technical report CMU-RI-TR-90-03, 1990.

[Lamit 81]
Louis Gary Lamit, Piping Systems Drafting and Design, Prentice Hall, 1981.

[Lander, Lesser, Connell 89]
Susan E. Lander, Victor R. Lesser, Margaret E. Connell, "Knowledge-Based Conflict Resolution for Cooperation among Expert Agents", *Computer Aided Cooperative Product Development MIT-JSME Workshop*, D. Sriram, R. Logcher, S. Fukuda, Springer Verlag, November 1989.

[Lander, Lesser, Connell 90]
Susan E. Lander, Victor R. Lesser, Margaret E. Connell, "Conflict Resolution Strategies for Cooperating Expert Agents", *Proceedings of the International Working Conference on Cooperating Knowledge Based Systems*, S.M. Deen, University of Keele, UK, October 1990.

[Londono, et.al. 89]
F.Londono, K.J.Cleetus, Y.V. Reddy, "A Blackboard Scheme for Cooperative Problem-Solving by Human Experts", *Computer-Aided Cooperative Product Development*, D.Sriram, R.Logcher, S. Fukuda, Proceedings of MIT-JSME Workshop MIT, Cambridge USA, Springer-Verlag, November 1989.

[MacCallum, 90]
K.J. MacCallum, "Does Intelligent CAD Exist?", *Artificial Intelligence in Engineering*,

Vol.5 No 2, Computational Mechanics Publications, April 1990.

[Madden, Pulford, Shadbolt 90]
Jim Madden, Carl Pulford, Nodel Shadbolt, "Plant Layout - Untouched by human hand?", The Chemical Engineer, 24 May 1990.

[Mandiau, Millot 92]
B. Mandiau, P. Millot, "Trends in Distributed Artificial Intelligence", *Artificial Intelligence Review*, Vol 6., No. 1, 1992.

[McIntosh 92]
Ken McIntosh, "Engineering Data Management", Technical File No 209, *Engineering*, April 1992.

[Motard, 89]
R. L. Motard, "Integrated Computer Aided Process Engineering", *Computers in Chemical Engineering*, Vol. 13, No. 11/12. pp 1199-1206, Pergamon Press, 1989

[Oliff 88]
Michael D. Oliff, "Expert Systems and Intelligent Manufacturing", *Proceedings of the second international conference on expert systems and the leading edge in production planning and control*, North-Holland, 1988.

[Polat, et.al. 93]
Faruk Polat, Shashi Shekhar, H. Altay Guvenir, "Distributed conflict resolution among cooperating expert systems", *Expert Systems*, Vol10, No4, November 1993.

[Pruitt 81]
Dean G. Pruitt, *Negotiation Behaviour*, Academic Press, 1981.

[Rase, Barrow 57]
Howard F. Rase, M. H. Barrow, Project Engineering of Process Plants, John Wiley & Sons, 1957.

[Reklaitis, Preston 89]
G.V. Reklaitis, K.L. Preston, "A perspective on computer integrated engineering", *Computer Integrated Process Engineering*, Hemisphar Publications, 1989.

[Rich, Knight 91]
Elaine Rich, Kevin Knight, *Artificial Intelligence*, 2nd Ed, McGraw-Hill, 1991.

[Rosenschein, Zlotkin 94]
Jeffrey S. Rosenschein, Gilad Zlotkin, "Designing conventions for automated negotiation", *AI Magazine*, Fall 1994.

[Russell, Norvig 95]
Stuart Russell, Peter Norvig, Artificial Intelligence - A Modern Approach. Prentice-Hall. 1995

[Rychener 88]
Michael D. Rychener, "Research in Expert Systems for Engineering Design". *Expert Systems for Engineering Design*, Michael D. Rychener. Academic Press, 1988

[Sage 90]
Andrew P.Sage, "Knowledge Support Systems and group decision technology", *IEEE*. 1990.

[SfK 92a]
SfK/BhR, ICI, Rolls Royce and Associates, Consultants, *Piping Reliability Steering Committee Meeting*, 11th November 1992.

[SfK 92b]
Don Miller, Steve Field, *SfK/BhR Visit Report*, Geoff Taylor, Laza Krstin, Alan Thornton, ICI Engineering, Chilton House, 25th June 1992.

[SfK 92c]
Steve Field, *SfK/BhR Visit Report*, Ron Palgrave, Chief Engineer, Ingersoll Rand Sales Company Ltd, Tyne and Wear, 15th October 1992.

[SfK 92d]
John Harrington, Keith Skilling, *SfK/BhR Visit Report*, Chris Wallsgrove, Brown & Root Braun, London, 15th December 1992.

[SfK 92e]
BHR, SFK, ICI, Rolls Royce and Associates, Consultant, "Minutes of the Piping Reliability Steering Committee Meeting", 11th November 1992.

[SfK 94]
John Harrington, Keith Skilling, *SfK Visit Report*, Roger Mallinson, ICI Billingham, 25th August 1994.

[Shaw, Gaines 1989]
Mildred L. G. Shaw, Brain R. Gaines, "Comparing conceptual structures: consensus, conflict, correspondence and contrast", Knowledge Acquisition, Academic Press, 1989.

[Simmons 93]
P.E.Simmons, "Stochastic decisions in engineering design", *Journal of Process Mechanical Engineering* Vol 207, IMechE, 1993.

[Steier, et.al. 93]
D.M. Steier, R.L. Lewis, J.F. Lehman, A.L. Zacherl, "Combining Multiple Knowledge Sources in an Integrated Intelligent System", *IEEE EXPERT*, June 1993.

[Sycara 89]
Katia P. Sycara, "Cooperative Negotiation in Concurrent Engineering Design", *Computer Aided Cooperative Product Development MIT-JSME Workshop*, D. Sriram, R. Logcher, S. Fukuda, Springer Verlag, November 1989.

[Talukdar, Fenves 89]
Sarosh N. Talukdar, Steven J. Fenves, "Towards a framework for Concurrent Design", *Computer-Aided Cooperative Product Development*, D.Sriram, R.Logcher, S. Fukuda, Proceedings of MIT-JSME Workshop MIT, Cambridge USA, Springer-Verlag, November 1989.

[Tomarchio 91]
A. Tomarchio, "Concurrent Engineering: Electronic Packaging Methodology Yields Quality Improvements", IBM Corporation, Manassas, VA, *AIAA Aircraft Design Systems and Operations Meeting*, Baltimore MD, AIAA-91-3151, Sepember 23rd, 1991.

[Tranfield, Smith 90]
David Tranfield, Stuart Smith, "Manageing Change - Creating Competitive Edge", ISBN 1- 85423-085-9, IFS Publications, 1990.

[Tribus 69]
Myron Tribus, *Rational Descriptions Decisions and Designs*, Pergamon Press, 1969.

[Uma, et.al. 93]
G. Uma, B.E. Prasad, O. Nalini Kumari, "Distributed Intelligent Systems: issues, perspectives and approaches", *Knowledge Based Systems*, Vol 6. No 2, June 1993.

[Winter 92]
Peter Winter, "Computer-Aided Process Engineering: The Evolution Continues", *Chemical Engineering Progress*, pp 76-83, February 1992.

[Wright 84]
George Wright, *Behaviourial Decision Theory: An Introduction*, Penguin, 1984.

[Yoshikawa, Gossard 89]
H. Yoshikawa, D. Gossard, *Intelligent CAD,I - Proceedings of the IFIP Workshop on Intelligent CAD, Boston 1987*, H. Yoshikawa, D. Gossard, North Holland, 1989.

Appendix A. The Process Engineering Design Function.

Overview

The process engineers are responsible for the design of a process and that it operates efficiently and safely under continually changing conditions.

The process engineers are usually the first people on the scene when a new contract is being discussed with the client. Their role at this stage is to weigh up the scale of the problem and make an estimate of the effort required. During most of the early stages in design the world evolves around process engineering, with all disciplines requiring initial design details in order to begin their work.

The output of the process engineering function consists mainly of the process flow diagram (PFD) complete with sets of heat and material balances and process specifications for all major items of equipment. This together with a variety of other documents is called the *process package*. It has been estimated that less than 10% of the total plant cost is devoted to this work and the decisions made at this stage account for over 80% of the total capital costs [Winter 92].

Process design has traditionally been viewed as a three phase function [Reklaitis, Preston 89];

i. Selection of the sequence of chemical and physical steps or unit operations to be employed to realise the synthesis path

ii. Assignment of equipment types to unit operations

iii. Definition of flowsheet structure, operating conditions, and functional equipment sizing.

A Basis for Design

Before these stages can proceed the process engineer must have an idea for the basis of the design. This may consist of the quality and quantity of the product; raw materials available and their characteristics; utilities and their temperatures and pressures; and details of by-products that may be required or produced. Problems concerning national standards on pollution and safety studies will also have to be researched [Field, Forsyth 92].

Block Diagrams

The process engineer uses the high level definition of the goals of the plant and produces a list of all the possible process routes available. These process routes are documented in the form of block diagrams. A block may constitute a major item of equipment or a section

of the process.

These block diagrams go through a continual refinement using energy and material balances until the engineer arrives at an acceptable design.

This process of continual refinement can be somewhat problematic. The process engineer is continually making decisions concerning the sequence of operations and deciding on the technology that should be used at each stage. He is responsible for deciding on a development strategy and errors at this stage can have dramatic knock on effects in later stages of design.

Initial block diagrams may be derived from existing processes that are known to be effective and are refined to suit the new requirements. Company policy may have a large input at this stage with regard to insurance. Insurance companies are wary of insuring new processes and design concepts because of the risk involved. This factor may hold engineers back from designing new and innovative process designs.

The concept of continual refinement of a process diagram is a powerful one, but not without its faults. An error with the initial process scheme will propagate throughout the detailed refinement and may not show up until late in the design. Also the use of previous process schemes may omit energy recovery, effluent treatment, by-product re-cycle, controllability analysis and safety analysis [Johns 92]. Continual refinement of a scheme can lock the engineer into an area of the design space from which it is difficult to break free, in mathematical terms, a *local optimum* is established.

The refinement of verification of block diagrams is performed until an acceptable energy and material balance is obtained and all major items of equipment have been identified. The final version of this diagram is referred to as a *Process Flow Diagram* (PFD) and is used as a reference by many disciplines together with the heat and material balance.

Process Flow Diagram

The process flow diagram as previously described shows the major items of equipment in a plant and depicts the overall flow sequence. Flows between the various items of equipment are referred to as *streams* which refer directly to a table identifying the *heat and material balance*.

The heat and material balance for a stream will show information such as mass flow, pressure, temperature, molecular weight and the various chemicals and their proportion in the stream. In a batch process where there is more than one discreet step, there may be several balance sheets that depict each stage.

The heat and material balance are produced by a steady state simulation of the process flow diagram. Packages available that perform this simulation (ex, ASPEN PLUS, CHEMCAD,

HYSIM) require libraries of models that can be assigned to each unit operation (ex, a heat exchanger). These models consist of a large set of non-linear equations that represent the performance of a particular unit. Certain key operating conditions may be set such as distillation column reflux ratios; pressure levels within gas/oil separation trains: pressure levels for refrigerant system; and stream-to-feed ratios in naphtha crackers [Kauders 80]. The flowsheet is simulated under control of an executive that is responsible for transferring the output of one unit operation to the input of another depending on the flowsheet structure. The executive is also responsible for dealing with re-cycles in the process and correcting for environmental conditions specified by the process engineer.

Besides detailed component libraries for unit operations, flowsheeting applications require an understanding of the physical properties of chemicals. These physical properties identify reaction sequences of chemicals under certain conditions (pressure, temperature etc) and various other details, for example if a chemical can be distilled. These physical properties libraries can be very large and there are international organisations (the IChemE in the UK) that collect and provide information on physical properties. It is very important that the physical properties are accurately defined, an error at this stage can lead to serious problems late in the design or even lead to failure of the project itself.

A major problem identified with current flowsheeting systems is that they are steady state, i.e. they only show the normal operating conditions. Brown & Root indicated startup conditions are not usually considered until very late in the life-cycle. It is not unusual to identify equipment that cannot cope with startup conditions and have to be re-sized at this later stage [SfK 92d]. ICI have cited a case when startup conditions were not properly accounted for and a section of a plant had to be dismantled because normal operating conditions could not be achieved [SfK 92e].

Current developments in equation-solving systems have paved the way for process engineering to undertake unsteady-state studies as part of their normal duties. This dynamic simulation of a process has a number of significant advantages [Winter 92];

* Startup/Shutdown analysis
* Hazard/Safety studies
* Operability studies
* Relief-system design
* Control schemes analysis
* Operator training

These advantages are likely to benefit reliability, improve control and generally lead to better designed and safer plants.

In addition to the major items of equipment and flow steams, a final PFD will also show the 'modes of control'. The modes of control are essentially the essential instrumentation needed to control the process. A standard set of instrument symbols have been defined by

the Instrument Society of America and these are usually followed to avoid confusion.

Process Data Sheets (PDS)

After completion of the process flow diagram the process engineer has to verify the scheme with a more detailed modelling of particular items of equipment[Winter 92]. Most plant items (including distillation columns!) are quite straight forward. However, the most common piece of equipment in a process plant is a Heat Exchanger which requires a more rigorous analysis of thermal operation. The results of these tests may mean that changes to the process design itself may have to be made.

The process data sheets are composed from the heat and material balances for all major items of equipment. These sheets can are then passed on to various other disciplines, with the process flow diagram, so that more detailed engineering can be completed.

Together with the process data sheets the process engineer may sketch a picture identifying the layout of the required system. This document is known as the *General Arrangement* diagram (GA). This sketch is usually required for pump installations where the elevations of equipment are required and rough approximations of distances.

Piping and Instrumentation Diagram

When the process plans have been completed and the construction contract has been agreed work can proceed on the Piping and Instrument diagram (P&ID). The P&ID is the next stage of refinement from the process flow diagram and shows a considerable amount of engineering detail.

The P&ID is a central resource of information for all design groups; electrical; piping; structural; instrument; equipment designers and so forth. All equipment (including spares), lines, instrumentation, valves, drains, utility lines etc are shown on the drawing. Standards for drawing the P&ID are specified at the start of a project in a legend sheet. This sheet is common to all disciplines so there is no mis-interpretation or confusion over the diagram. The legend sheet symbols may be specified by the client at the start of the project so all records are maintained in a standard format. If the client does not insist on a particular notation then the contractor is likely to select a legend sheet that he is used to working with. However, there are usually no major differences between different notations. The instrument society of America has specified specific notation that has been adopted in most designs.

The P&ID will indicate detail that is essential to the process design. For example, a pump has been specified that requires a positive NPSH so elevation detail may be necessary. If specific detail has been left out of the drawing then the relevant disciplines can assume that the detail was not important and they are free to specify a configuration that suits their requirements. Problems may occur at this stage as the process engineer may not be aware

of detail that is important to the other disciplines. Also the process engineer may specify specific constraints on the process diagram that can cause unnecessary complications further in the life-cycle.

The process engineer must also estimate line sizes at this stage. The following criteria must be considered when determining the line size [Kauders 80]:

i. Economic factors (a balance between the line size and compressor or pump horsepower)
ii. Velocity limits due to erosion, corrosion or noise and
iii. The available pressure drop (e.g, to meet pump required NPSH)

The P&ID, like the PFD is a steady state diagram. Startup and shutdown conditions are not indicated on the diagram. Emphasis on startup and shutdown connections (e.g. a flare or drain) should be made on the PFD and the relevant detail added at the detailed design of the P&ID.

Appendix B. Piping design - principles and problems

Visit Report date: 15th February 1994
People involved: Barry, Piping and Material Engineer
(the name has been changed to maintain anonymity)
John Harrington

Objective: To interview a piping and mechanical engineer with the intent of understanding more about piping and material aspects in chemical plant design and to understand design problems.

Summary

Barry is a piping and materials engineer by trade but has had various experience with other design departments. The aim of the day was to understand the work of the piping group in general terms and understand his views on the whole design process. Barry has a little process knowledge and made his 'best guess' at most of the questions asked of him. Be aware that information regarding design in general is not exacting.

We did not discuss the specifics of any particular company, he took a general view of design from the many companies he has worked in. The noticeable differences are in the format of the documents that are produced -although the procedures and knowledge applied is mostly the same.

Barry is realistic, very much more down to the detail of how things will fit together. This is unlike the more theoretical and 'model' like world of the process engineer. He job is mainly routine, in his words 'to interpret standards'. KBS technology is therefore suitable in this type of domain.

The process engineers are chemical engineers and should have some knowledge of piping, vessels, and later design aspects. He is often surprised at how much the older process engineers know about the later stages of design. These process guys are involved throughout the entire design, involved in writing the operating manual and in commissioning.

The process engineers produce the PFD. The PFD defines the design conditions, major equipment items, elevations (e.g. tower discharge lines) etc. The completion of this diagram starts off three concurrent tasks;

1/ P&ID development
2/ production of Process Data Sheets
and 3/ the production of the plot plan.

The *systems man* is responsible for managing the development of the P&ID. He assigns the line sizes. He is usually a member of the process development group, and therefore works amicably with the process engineers. The systems guy also provides the link between the process and mechanical design departments.

The systems guy produces the first draft of the P&ID. Barry did not have any concept of the various stages of the P&ID -it was an iterative task and there were around ten releases of this document during the life of a project.

From the first draft of the P&ID the plot plan is produced. The plot plan is produced and controlled by the piping engineer, although it is really a group effort with the systems and process engineer. He finds that the systems engineer is easier to deal with so they are predominantly involved.

The plot plan is derived by placing the major items of equipment noted on the process flow diagram onto a designated plot. The plot is (usually) specified by the client, i.e. 'you have to fit the plant into that space'. You have to consider any existing roads and utilities available on the site. The design of the layout appears to be particular to the individuals style. An example was described where he produced a plot layout for a client -first he had to change it for his manager, then this was modified again by the project manager, and finally the client changed it back to how it was initially. Barrys style was to place the pipes around the walls of a building to enable easy access and avoid accidents, although the manager & project managers view was different. There is no scientific method of identifying a 'best design'.
When specifying the layout there are particular variables that you wish to minimise. For safety reasons you have to ensure that particular items of equipment are not too close to each other. Reasons for having equipment items close to each other are to avoid long pipe runs when the pipe is carrying hot or cold fluids. You can obviously loose significant amounts of energy in long pipe runs. Noise is also a consideration when placing machinery. In order to evaluate the layout it is necessary to work closely with the process.

systems engineers, and the client in order to make a good decision. The client normally has a fair amount of input.

The P&ID and the plot plan are evolving at the same time. One of the important jobs by the Systems guy is to assign line sizes to the piping. This line size is dependent upon the plot plan because of the pressure drop, the number of elbows, tee's and valves etc. Changes to the plot plan at any stage during development may seriously effect line size, which in turn can effect any fittings required for the line -there is considerable iteration and discussion throughout these early stages.

There is no consideration given to the pump in the selection of pipe sizes (long term costs etc.). Barry says this is an ideal but has never seen this done. The client wants 'everything now' so time is critical, and the attitude is to accept the cheapest that is 'fit for purpose'. If you know a vendor that is cheaper than another, although his service is very poor, your project manager will make you go for the cheapest -the 'fit for purpose' rule overrides consideration to long term plant operation. Barry did not want to discuss pumps in any further detail because he really didn't know much about them.

The process engineer is responsible for the PFD. This document (at it's final version) contains all the equipment items (*including* valves, pumps, and obviously major equipment items) that are required to realise the process. The additional valves, drains, check valves etc are added to the P&ID by the systems guy when they are required for utility, safety and maintenance purposes. (Note the important distinction between the two documents, the equipment to realise the process = PFD, PFD + utility + safety + maintenance = P&ID). The process group will also specify where valves are to be placed on a pipe -this shows their involvement throughout the complete design process.

The systems guy's job is routine and he follows a strict set of rules, set up for every job, when defining the auxiliary safety and maintenance equipment. These rules are generally the same throughout all design projects although different companies have minor differences. These rules cover things like, *always place a check valve on the discharge of a centrifugal pump*, *have bypass lines around pumps* etc. Any rules that are not applied lead to changes at a later date. It is interesting to note that this is a *routine* process and that the systems guy should not deviate from these rules!

At the same time the PFD is generated the process data sheets (PDS) are produced for the major items of equipment. Usually these types of equipment have long lead times and therefore may have to be ordered ASAP, or even before the PFD is completed!. Equipment with long lead times are things such as *towers* and possibly *heat exchangers*. In the case of towers, the process engineer has to specify the size of the lines connected to the tower very early. The systems guy will strive to achieve this line size although it may not be possible. This may lead to changes in an order, which can be expensive. The process engineer will identify certain elevations on the PFD where appropriate. For things like towers, he will specify where the different lines should be connected to the tower (using

0 height as the base) as these heights are important for the process. The process data sheets are issued to the various disciplines, vessels, rotating equipment, heat exchangers, instruments etc.

The instrument group make life difficult for piping, they are usually *first on, and last off* the project. They have become considerably more important since computer control systems came on the scene.

Barry had a lot to say about towers, this is obviously the most demanding equipment item for him and is obviously an important item for realising the process. When designing a tower the vessel guy needs to define things like ladders and platforms. When it comes to the orientation of the tower, he has to talk to the piping guy to identify where pipes should be connected -you don't want to be routing pipes around a vessel just to make a connection. When designing the tower you have to consider the wind around the tower, the bolts required for a base and consider the pipe stressing around the tower. He cited a case recently of a tower that was moving ~5 foot in a wind due to incorrectly specified bolts -you can imagine the stress on the connected pipes! After the Vessel guy has completed his drawing it is sent to a manufacturer who produces a manufacturing drawing.

The systems guy maintains control of P&ID development. When the systems guy defines additional valves for maintenance/safety, or when he spots a problem he discusses the design with the process engineer. If valves are required the process engineer would complete a process data sheet.

After the piping engineer has completed the layout he develops the general arrangement diagram of piping and vessels. A number of studies are performed on 'scrap paper' to layout piping, items of equipment, instruments etc, and these are pulled together in a formal general arrangement diagram. The isometric drawings for each pipe (or part of pipe if long) can be taken out from the general arrangement diagram.

At the early stage of P&ID development and when the plot layout is available an hydraulic analysis is performed on the pipe to aid in determining size. Subsequent changes and extensions are not subject to an hydraulic analysis. Barry once found a piping engineer who by default always specified one pipe diameter bends. Barry realises that this increases the pressure drop and attempts to avoid these bends, usually going for 1.5 diameter bends. There appears to be significant over design to compensate for these types of problem. Barry noted one case where a tower had been delivered with half its plates missing. Due to the time constraints they performed a analysis and found that the plant could operate with only half the trays anyway and so the plant was commissioned!

Barry summarised his job as to *interpret standards*. There are around eight standards for the design of plants, the most important one to him, his base document, is B31.3 -for onshore chemical plants. This standard is specified by the client at the start of the project. He thinks that one has come out recently specifically for offshore plants. This document

is really a 'standard for standards'. It refers you to the appropriate standards and design codes to use at the appropriate design stages. It tells you WHAT you should do, rather than the HOW. You NEVER deviate from these standards unless you have a really good reason. If you do deviate from these standards you have to discuss the reason with your manager, the client and ultimately negotiate with the insurance company.

When the PFD is issued, the senior piping and material engineer specifies the pipe classes for the project in the *Piping Materials Specification*. This specification is used by all the piping engineers throughout the project for designing and specifying pipes, pipe connections and instruments on the pipe, joints, bends etc. There may be between 10 and 40 pipe classes specified for a project depending on the complexity and size of a project. When a pipe is specified, the pipe number should identify the appropriate pipe class that the piping engineer has designed the pipe to. The pipe class restricts the engineers to selecting specific type of instruments and materials, such as check valves and connections (e.g. flange, butt welded).

The pipe classes are reviewed by the materials specialist, the process engineer and the systems engineer. Initially the metallurgist gives a material for each stream although this is quite preliminary. Apparently the materials guy is really difficult to talk to, you can never get a straight answer out of him. All the piping engineer wishes to know is the correct material to use, but the materials specialist always has another hand, e.g. "CS is ok, but on the other hand...." -really annoying.

The detailed piping general arrangement is done on CAD. The ISO's are taken directly from this drawing and a materials take of list produced for that line, this includes things like valves, pipe lengths, bends, flanges etc. CAD is very useful for obtaining a final take off list automatically. However, by this stage most things are on order and the only purpose it serves is to check the order was correct. A material takeoff (MTO) is usually performed in three stages, a preliminary MTO, an intermediate MTO and a final MTO.

A preliminary MTO is done by a piping engineer when he has the plot plan and early P&ID. The piping group usually has an internal MTO group who are responsible for this. The estimate is taken using a slide rule against the P&ID and plot plan. He often has to estimate pipe fittings, flanges, gaskets etc. This preliminary MTO provides the basis on which to buy pipe. The study is usually between 60 to 70% accurate and there is a fair chance that some pipe sizes will change. From this MTO the piping engineer writes the requisition and this go's out for bids.

The intermediate MTO is performed when more information is available, the engineer now has some GA's and he should use these drawings. The estimate will now be around 80-90% accurate. The engineer will then fax the vendors and purchase on this intermediate takeoff. It is not often, but you may purchase on the preliminary takeoff -you can obviously make expensive mistakes.

The final MTO is performed when all the ISO's are available. He checks the final list against what has been ordered. Any differences and he must order further supplies. The final MTO is valuable and is the only one produced by CAD, this is unfortunately a long way down the line.

In offshore duties a weight MTO is also performed. This ensures that the weight is appropriately distributed around the platform to ensure the centre of gravity is in the centre and the platform does not topple. Barry noted one case where a preliminary MTO was performed without detailed knowledge of what valves were to be used. In this case the engineer specified Alloy 825 valves as the example because he considered the final valves to be selected would have a similar weight. Management however were in a rush to complete things and they purchased £1 million of Alloy 825 valves on the preliminary MTO that were un-usable!

The engineers before purchasing equipment have to liaise with the construction planning department who ensure that things arrive on the site at the correct time. This is another pressure on the engineer if the item is major and is the first item required to be installed.

Pipes are ordered in *random lengths*. The definition of these random lengths are specified as a standard as an allowable tolerance for the manufacturers cutting machines. For example, if you order a 5 metre pipe, the length delivered can be between 4 and 7 metres long! Obviously this tolerance can be improved through the improved cutting technology, although the pipe suppliers do not make use of this to benefit the clients. On a previous project Barry ordered 961 meters of pipe, and was delivered and charged for 1031 meters. The manufacturers will provide near to the maximum limit with which they can get away with in order to improve profit. Additional to this, the piping engineer has to allow for a 10% cutting tolerance. For example, if you ask stores for a 11m pipe, and they pull a 12m pipe out of stock, they will chop off 1m and this is likely to be wasted. If there are long pipe runs, you will order pipe lengths as long as possible. There are limits in size for reasons of transportation (~20m?), but the customer does not want to see too many line breaks in a long pipe.

Piping stress is a detailed subject and we did not have time to go into any detail. Barry did mention the importance of distance between supports in relation to the elasticity limits of the material. If the supports are too far apart, the pipe may sag in the middle due to the weight of the material (there is a maximum weight where pipe sag will occur). At higher pressures the material elasticity becomes an irrelevant consideration due to the thickness of material required to overcome the pressure.

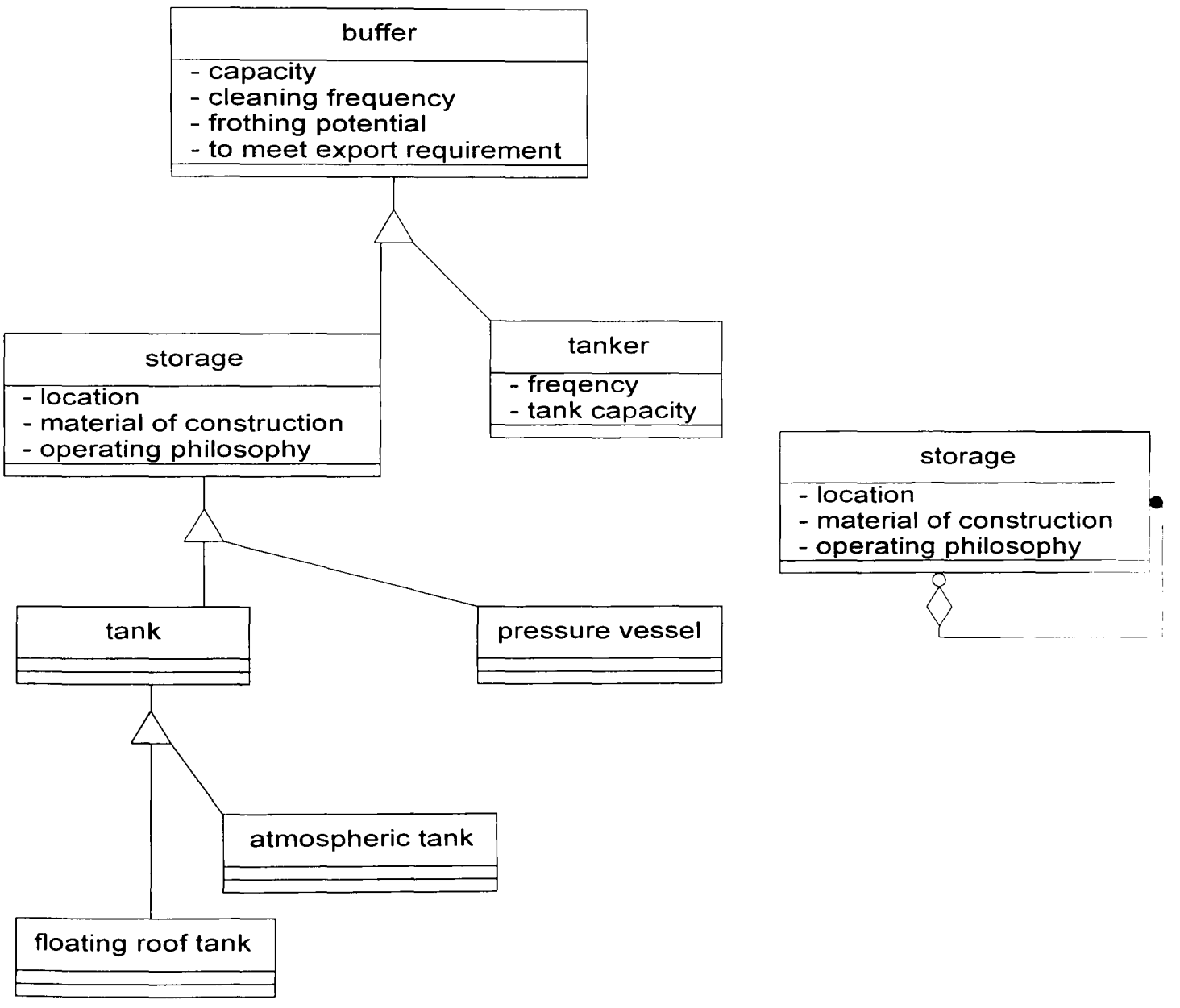
The lack of integration and knowledge flow between design and operating disciplines was exemplified when Barry did a job for BP. The design phase included a number of experienced operating personnel. This was the first time that Barry had worked with operating people and was surprised by the different attitude they took to design. He cited an example of a device to restrict flow in a pipe. The operating engineers selected a

completely different device for the duty due to the fact that the contract maintenance staff could not be trusted to close the device without damage. Apparently contract maintenance staff are not too concerned with the quality of their work and are quite prepared to fudge problems. This example highlights the need for the operating engineers to be involved at an earlier stage in the design process and for operational feedback to the design team.

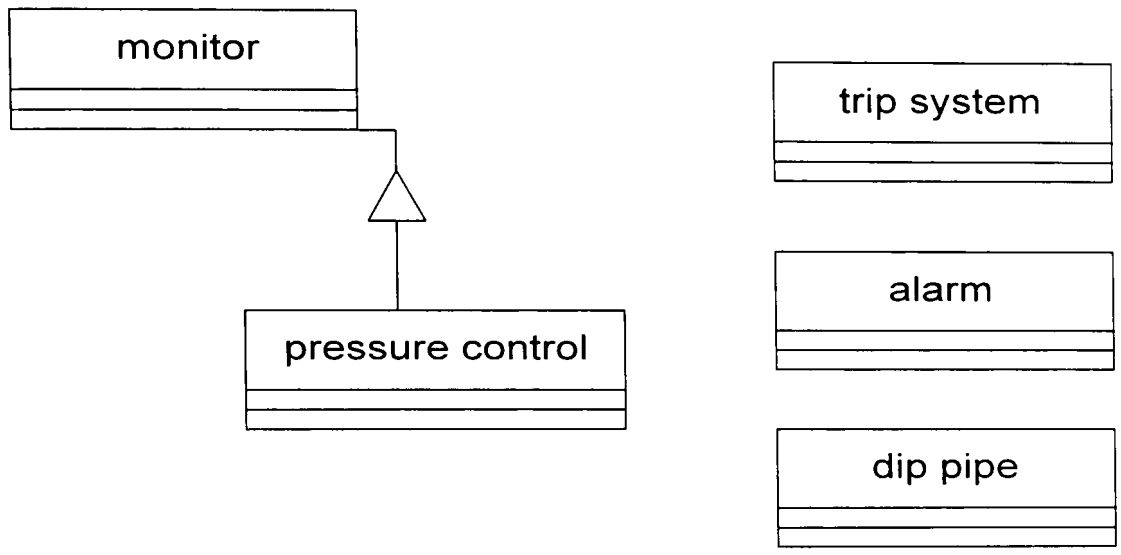
Appendix C. Scenario Object Diagrams.

Buffer

A buffer is a temporary storage place for product. A split has been made between storage (which includes tanks and pressure vessels) and a tanker, which is the object that signifies a road tanker. The diagram shows that a storage mechanism can be related to many other storage mechanisms, in this particular design case a storage mechanism was split into a requirement for two storage mechanisms for reliability.



Control + various



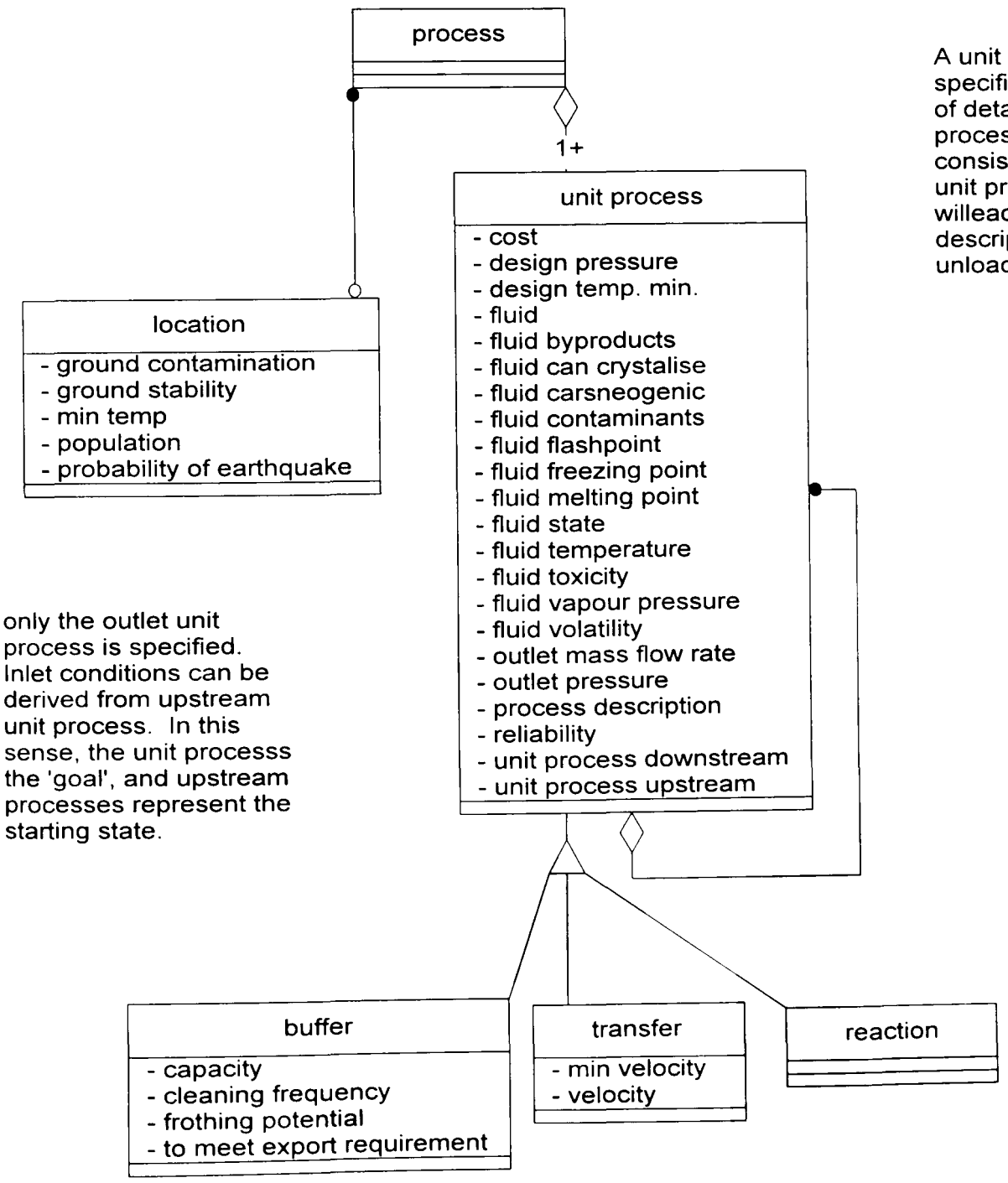
The above object model depicts the control elements in the design case.

Process

A UNIT PROCESS represents a process unit on the process flow diagram. The UNIT PROCESS can be specialised into one of the more specific processes - buffer, transfer or reaction. It has been assumed for this case example that these three basic processes will cover all the different processes in our example. Only the process conditions on the outlet are specified in the unit process. Inlet conditions can be derived from upstream unit process. In this sense, the unit process is the 'goal', and upstream processes represent the starting state. All chemical properties regard the outlet state, not stored state of the fluid in the unit process.

The TRANSFER object depicts a transfer process. A transfer process is any process that is involved explicitly in the transfer of fluid (e.g. stream, pump, control valve).

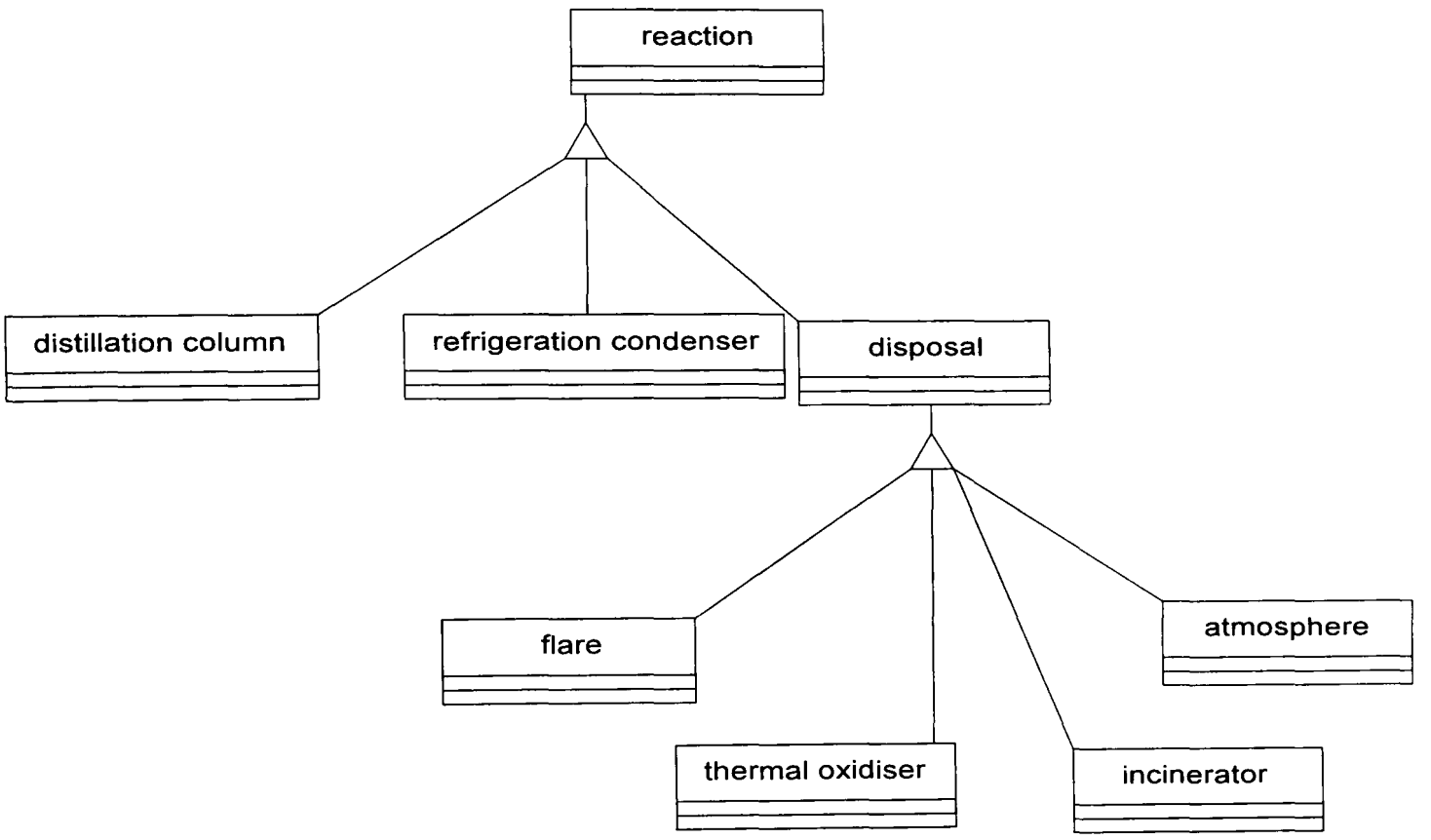
A BUFFER is a temporary storage place for feed/product (e.g. tank, pressure vessel).



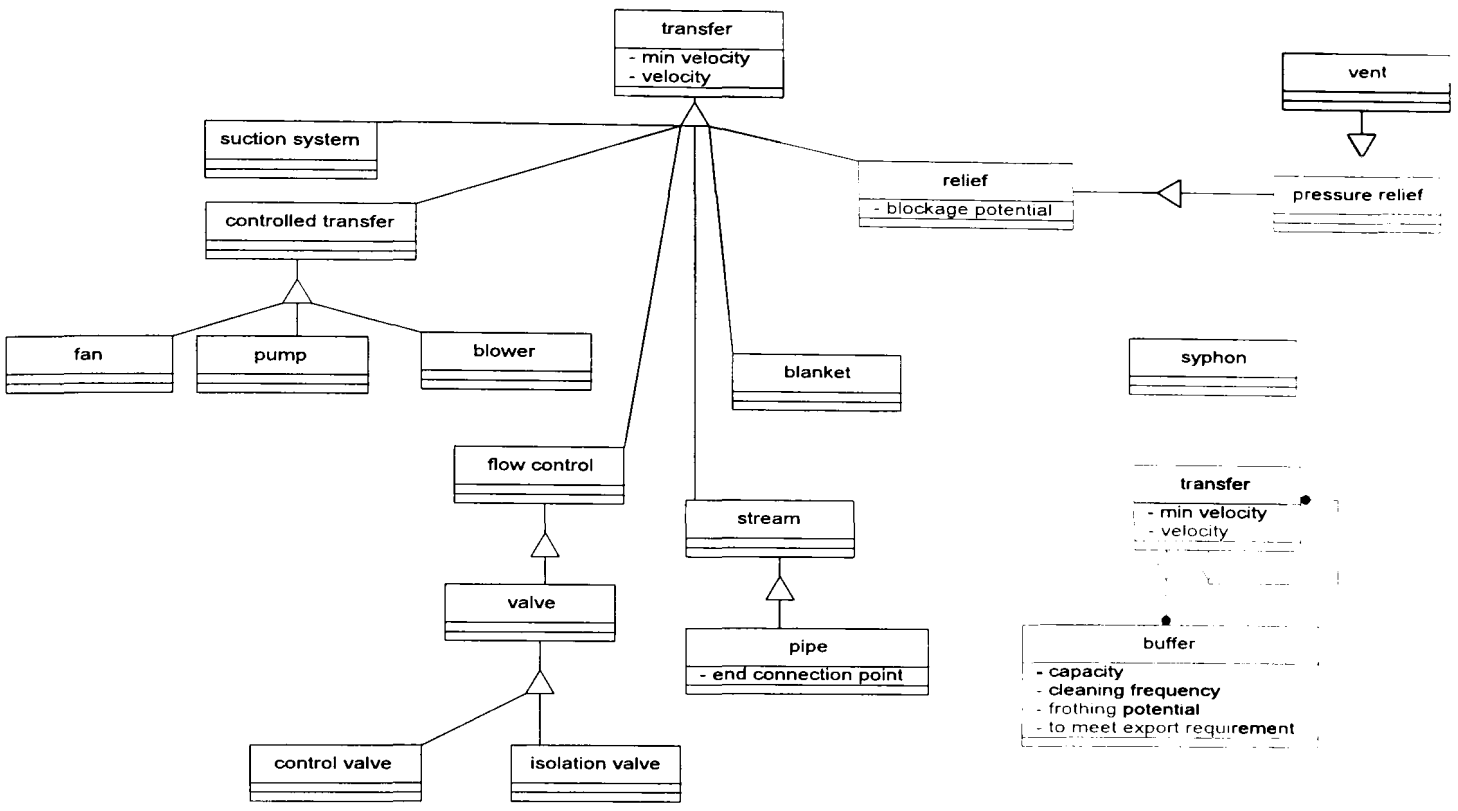
The REACTION represents all devices that modify the chemical characteristics of a fluid (e.g. distillation column, reactor).

Reaction

The types of reaction above represent only those objects identified in the design case. A REACTION is any device that changes the chemical properties of a fluid.

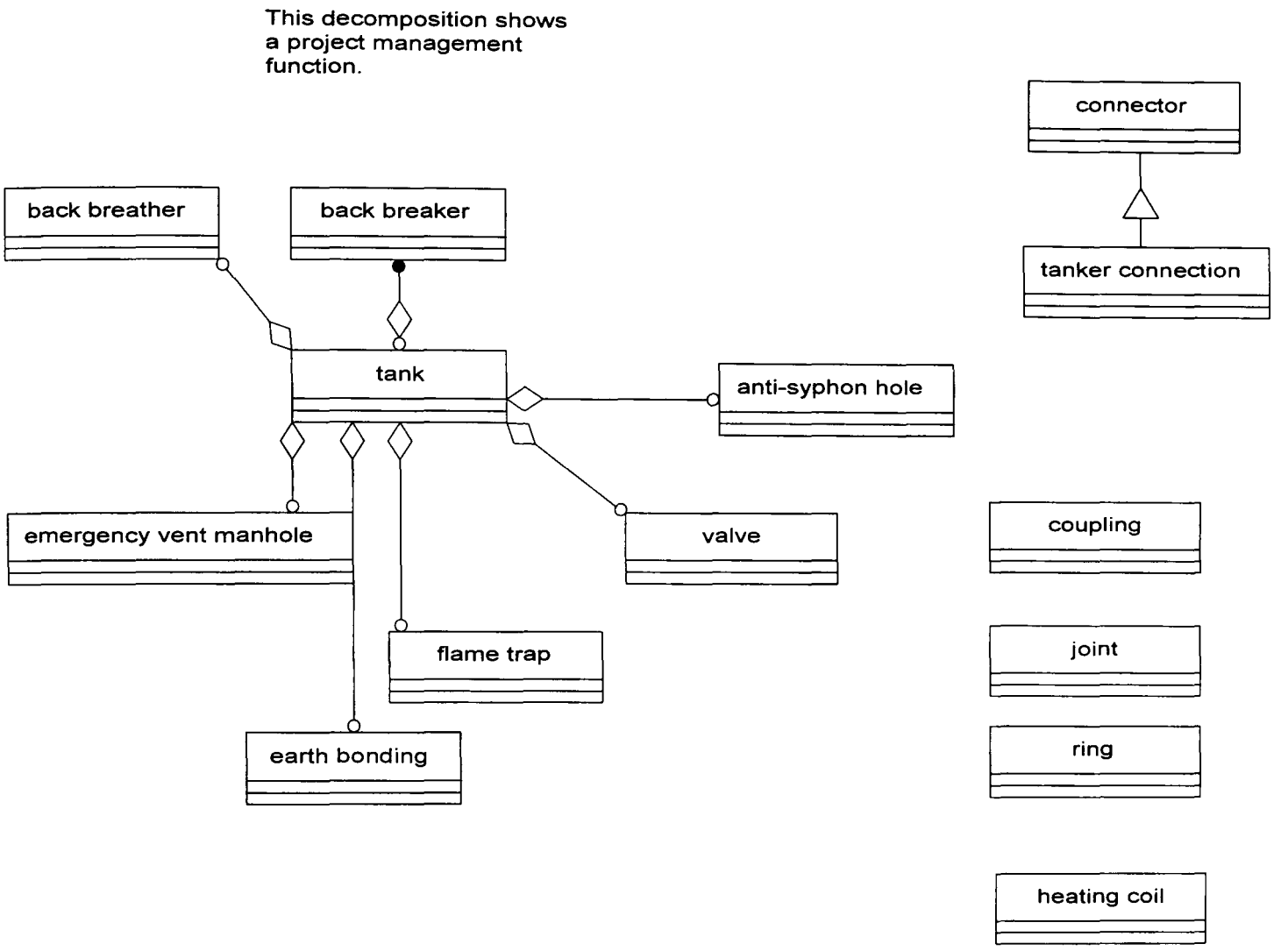


Transfer



The above diagram depicts all the types of TRANSFER process that were identified in the design case. These devices have been broken down into the groups where the transfer process is controlled by devices that increase the flow (CONTROLLED TRANSFER), where flow is controlled through a restriction (FLOW CONTROL), where flow is for relief purposes (RELIEF). Various other types are shown here that were identified in the design case, but they have not been characterised here as they are single cases and a classification would not help.

Tank System



The above object model depicts items that may be connected to a tank. The fact that the items are connected to tank does not signify a prescriptive approach to where items should be specified. The diagram however does depict physical devices that are not covered under the unit process headings such as the coulping, joint, ring etc.

Appendix D. Objects in the design case.

air (P)
alarm
anti-syphon hole
atmosphere
atmospheric tank
authority (OS)
back breather
back-breaker
benzene (P)
benzene column (P)
bird (OS)
blower
blown tanker (IND)
chemical (T)
CIMA regulations (OS)
concrete (OS)
connection (T)
connector
control valve
coupling
cracker (T)
dip pipe
distillation column
earth bonding
earthing (T)
ELD (OS)
emergency vent manhole
facility (T)
fan
feed tank (IND)
flame trap
flame (OS)
flare
flare stack (T)
floating roof tank
floating roof vessel (T)
floor (T)
flow control
fluid
gas (P)
ground (T)
halogen (P)
heating coil
incinerator
inlet vessel (P)
installation (T)
inventory (OS)
isolation valve
joint
kit (T)
liquid (P)
location
material (P)
material data sheet (OS)
mild steel (P)
mixture (T)
money (OS)
nitrogen (P)
nitrogen pressure control (P)
nitrogen blanket (P)
earthing electrodes (T)
off-site (T)
organic (P)
oxidiser (T)
person (OS)
PFD (OS)
pipe
pipeline (T)
plant (T)
population (OS)
pressure control
pressure vessel
process facility (T)
product (T)
pump
re-cycle (T)
recycle loop (IND)
refigeration condenser
regulation (OS)
relief
ring
road traffic (T)
safety data sheet (OS)
sealed coupling
sludge (P)
solids (P)
still (T)
stock (T)
storage tank (T)
storage
suction system
syphon
system (OS)
tank
tanker
tanker connection
thermal oxidiser
toluene (P)
trip system
trip (T)
trip valve (T)
valve
vapour (P)
vent
vent breather (T)
vent valve (T)
vent header (OS)
VOC (P)
water (P)
works (OS)

for inclusion in the design environment:

- remove objects which are really attributes of objects rather than objects themselves. e.g. 'air' can be a value of 'fluid description' of object fluid ***(P)***

- only include objects which we are required to

model for the plant design, and which we are interested. Do not include concepts outside the system ***(OS).***

- used under another term. Example, fluid is used instead of chemical ***(T).***

- remove objects that are modelled in direct ways. e.g. a recycle loop is a pipe from the plant back to a tank ***(IND).***

Objects Modelled in CDEX for design case

alarm
anti-syphon hole
atmosphere
atmospheric tank
back breather
back-breaker
blanket
blower
connector
control valve
coupling
dip pipe
distillation column
earth bonding
emergency vent manhole
fan
flame trap
flare
floating roof tank
flow control
fluid
heating coil
incinerator
isolation valve
joint
location
pipe
pressure control
pressure vessel
pump
refigeration condenser
relief
ring
storage
suction system
syphon
tank
tanker
tanker connection
thermal oxidiser
trip system
valve
vent

Identify higher level classes that have not been identified but are inherent in the discussion. e.g. 'storage' is a unit process.

buffer
controlled transfer
disposal
monitor
pressure relief
process
reaction
stream
transfer
unit process

remove objects that could be attributes of the objects identified.

removed:
fluid

Properties of identified design objects from the design case

Variable	**used in context**
cost	tanker, storage, pipeline
reliability	supply (transfer)
mass flow rate	plant requirement
capacity	vessel storage
chemical mix	ground contamination (location)
population	location
fluid	process
temperature	process
flashpoint	process
design pressure	tank, pressure vessel, atmospheric tank
operating pressure	tank
can block	vent
can crystalise	process fluid
melting point	process fluid
freezing point	process fluid
min temperature	location, vessel
volatility	process fluid
toxicity	process fluid
carsneogenic	process fluid
number per day	road tankers
cleaning frequency	solids in storage
max storage capac.	CIMA/regulations
max oper. temp.	process fluid
contamination	process fluid
fluid state	liquid, gas, etc.
material	construction materials, tank etc
velocity	fluid transfer
connection point	where pipe connects to vessel (top, base etc)
frothing possible	process fluid when loading tank

Appendix E. Objectives in the design case.

The following is a list of what can be construed as goals derived from the vessel design scenario discussion. They are listed in the order that they appear. Some goals are repeated, but we will derive an understanding of the importance which engineers subscribe to particular goals.

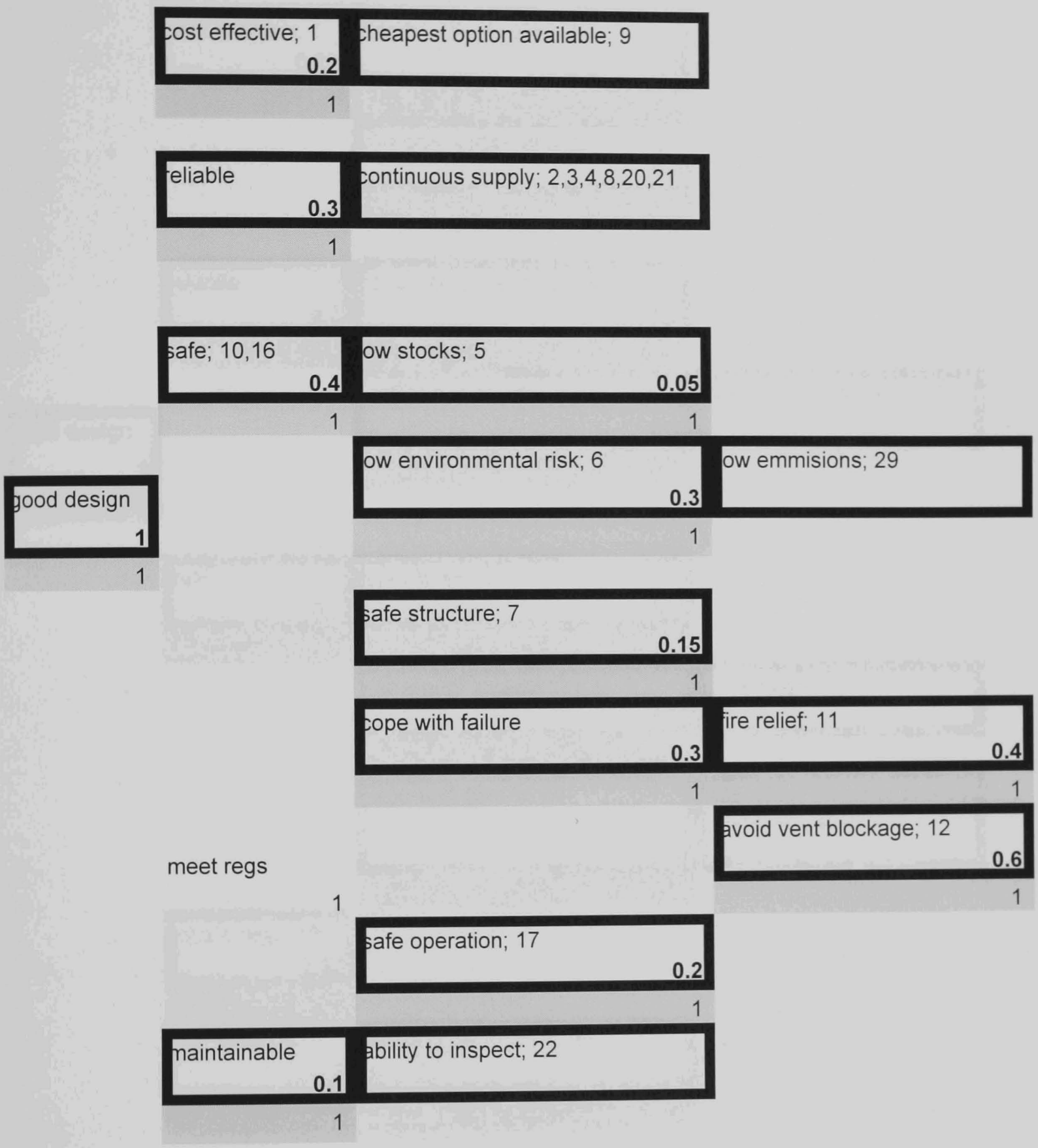
Note that there is some overlap with the Actions and Requirements list which is expected. Each objective is tagged with the engineer who showed the concern for a particular objective. [R] for Roger and [J] for Jeremy.

1 [R][J]	everyone is cost conscious at this time
2 [R][J]	reliable continuous supply
3 [R][J]	continue production
4 [R][J]	need enough inventory to keep plant running for two days
5 [R][J]	it's best to reduce stocks
6 [R][J]	ground contamination
7 [J]	securing vessel
8 [J]	polymerise
9 [J]	cost
10 [J]	inherently safe
11 [J]	deal with fire relief
12 [J]	no blockage of vent
13 [R]	maintain minimum temperature in vessel
14 [R]	cost
15 [R]	standard items of kit
16 [J]	harardous
17 [J]	do not want people to panic
18 [R]	regulatory requirements
19 [R]	cost
20 [J]	road tankers to discharge
21 [J]	availability
22 [J]	inspection
23 [R]	less chance of overfill
24 [R]	cope with emergency conditions
25 [R]	loss of thermal oxidiser
26 [R]	reverse flow from benzene column
27 [R]	loss of nitrogen pressure control
28 [R]	disconnect system safely
29 [J]	reduce the emisions

The following two tables show the weightings attributed to each of the engineers considering this design scenario. The most general objective is on the left of the table. Each objective is tagged with a reference to the list of objectives above. The value in the

box, below the objective description, is the weighting or 'contribution' that the objective has in meeting its parent objective. For example, cost effective has a contribution of 0.2 to a good pump system for Jeremy. The value below the box (always 1 in these cases) is the value determined from analysis of the proposal, it is a normalised value derived from some measurement in the problem domain. If an objective has sub-objectives, then this value is the contribution of all the sub-objectives towards meeting the appropriate objective.

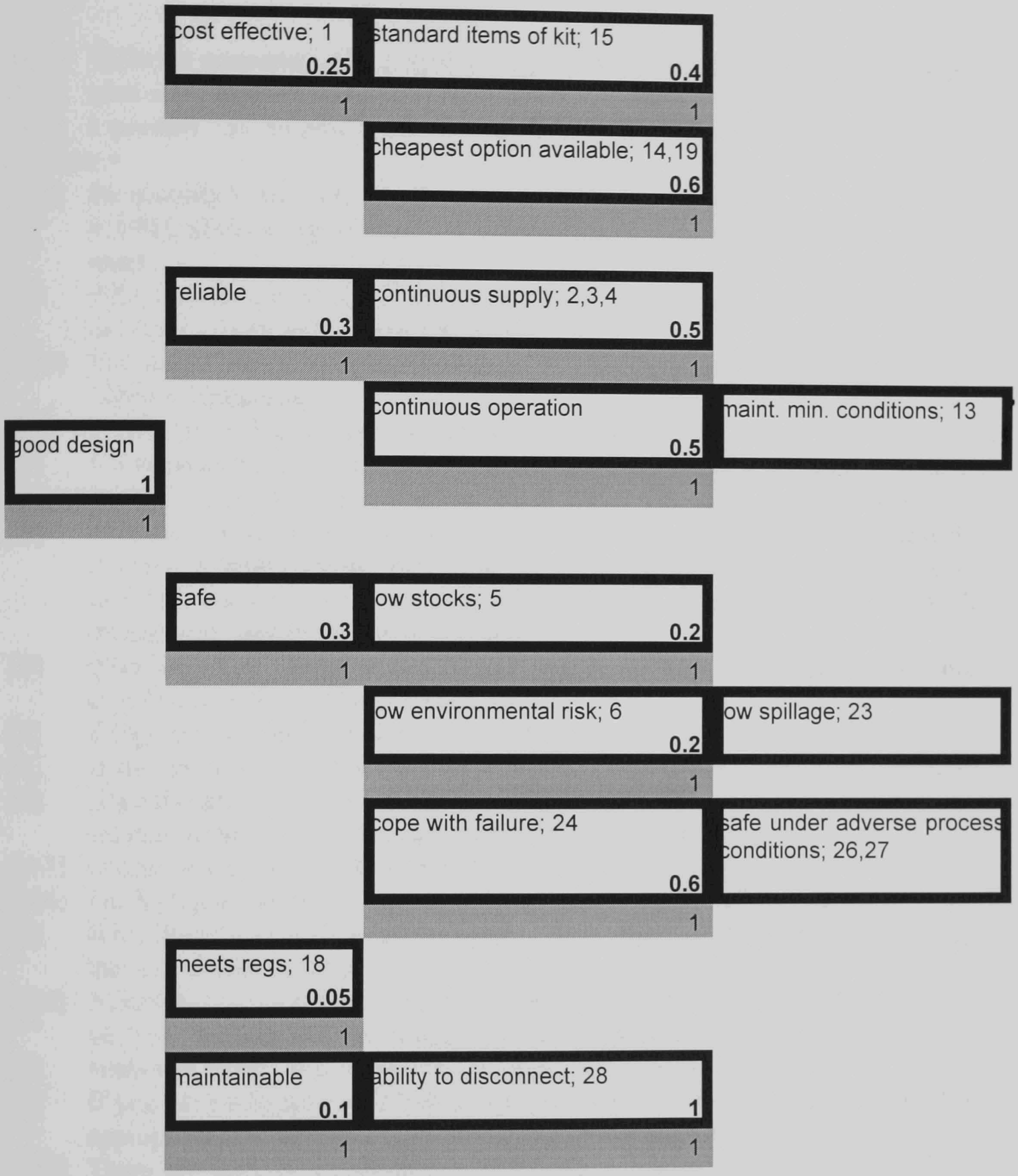
Jeremys Objective Hierachy



Notes:

Jeremy did not identify regulations as a distinguishing factor in determining a design approach in the case scenario.

Rogers Objective Hierachy



Appendix F. Causal relationships in the design case

[RM] We're not connected up to a pipeline (and therefore no reliable continuous supply)
[RM] need some means to continue production if deliveries can't be made
[RM] a product can be purchased and stored at a time of year when it is cheap to purchase.
[RM] the quantity to store provides an idea on the size of the facility
[JI] (COSH, SIMA etc) guidelines will determine if you can store a product, and how much.
[JI] if the toluene was coming back hot, and had a vapour pressure of 3 bar - you can not store it in an atmospheric pressure tank
[RM] You would want to know the flashpoint - in order to determine type of flame traps, nitrogen blanketing
[JI] a thermal oxidiser presents a back pressure problem
[JI] It may well be cheaper instead of putting fans in and all the problems with that, to design it to be a pressure vessel and blow vapour through to an incinerator
[JI] with an existing tank that is designed up to 18inch water gauge you can't blow the gas directly through to the incinerators so you would have to put a suction system in which can draw air in which would {mix with air and} create a flammable atmosphere, and create flammable headers
[JI] if we know that it has to go to a thermal oxidiser we would be designing a tank that could vent easily to thermal oxidiser
[JI] design pressure reflects back on the type of tank
[JI] If we have nitrogen blanketing we would still have to vent back to an incinerator
[JI] If we thought that in the long term we would never connect this to an incinerator we may settle for an atmospheric tank, but then vent to a flare
[RM] overflows can only work on atmospheric storage tanks.
[RM] For high pressure tanks you would have high pressure alarms or reliefs
[JI] If the fluid can crystallise, and to design it to be inherently safe you have to design the tank to take the dead end pressure of what can be feed to it
[RM] Atmospheric information is around, and this data would identify whether we would need any heating coil to maintain a minimum temperature in the vessel.
[JI] vents will block - this effects the design pressure rating
[JI] If you have a floating roof tank you wouldn't have to worry about venting. It is appropriate in situations of low volatility and low toxicity
[RM] There are distinct disadvantages of having a storage tank on site with large quantities, as well as the cost
[RM] we have a requirement for 10 tonnes an hour, then you can't meet this without storage
[RM] If you have to take it down every two years to clean out the sludge, you have to have two tanks minimum
[RM] if you have a problem with one tank because its leaking, you have another one to draw from

[RM] two tanks with one left empty so that if you have a leak you can leak into the empty tank - this has happened in some cases.
[RM] Another problem is cross contamination - say one person loads it up with another fluid which is not toluene
[JI] if the engineers on the ground had the wrong connections they would change them
[RM] We need a nice large tank - keep it topped up so that people don't panic about connecting the tanker pipes within 5 minutes for example.
[JI] contamination and benzene effects fluid flammability
[JI] contamination and benzene in the process effects the flammability
[JI] contamination may be a function of re-cycle - rather than actual feed
[JI] we're not allowed to vent toluene to atmosphere
[JI] a pressure vessel does not stop us having vents
[RM] pressure vessel does stop us having vents - the only thing that does is a floating roof vessel.
[JI] but we can't have a floating roof vessel because of the high volatility and toxicity
[RM] we need a vent because it is an atmospheric tank
[RM] need a thermal oxidiser because of regulatory requirements.
[RM] reluctant to say we should put a pressure vessel in their because of cost, its a big tank & costs a lot of money.
[JI] we could have a suction system, however we now have the possibility of the suction system sucking your tank in.
[RM] yes, an atmospheric tank won't withstand suction.
[JI] back-breakers, that blow when the tank is under suction
[JI] flammable vapour coming through which can be diluted with air to make a flammable mixture
[JI] something going towards the thermal oxidiser which is flammable mixture, which can cause detonations and explosions in the vent header
[JI] You then have to have vent valves to vent to atmosphere or a flair stack
[JI] if that vent valve is open when it shouldn't be open, then you can suck in air through that and therefore have a flammable atmosphere
[JI] if you have the tank as a pressure vessel, and the vessel had more than a couple of bars in then you won't get the round tankers to discharge
[JI] method of filling - lets assume blown tankers (pressure on the top and blow it out) at say 2 bar gauge.
[RM] empty them, I would of thought pumping was appropriate
[JI] well it would be air from a tanker because they haven't got nitrogen on the tanker have they
[JI] when its empty, you have all this pressure with an open end through the pipe, up into the storage and out through the vent.
[RM] well the way to prevent pressure going through the pipe up to storage is through a trip valve.
[JI] trip valves do not always work, it may jam open
[JI] a problem with vents is keeping them free
[RM] keeping vents free is not a problem as there is no polymerisation

[RM] vents can get blocked by birds nesting in the vents
[JI] no problem with birds nesting in the vents, this can be designed against
[JI] The oxidiser works under controlled conditions with guaranteed residence times. We can guarantee to kill all the halogens
[JI] with a flare stack all you are trying to do is get a reasonable amount of combustion
[RM] liquid is pumped from the tanker, therefore we won't get gas breakthrough from the road tanker
[JI] if we did loose level from still and blow through, we would end up with 6-8 m cubed an hour of vapour from the still (distillation column)
[RM] minimum temperature is 0 degC. We therefore don't need heating coils and because of hot re-cycle back from plant
[JI] solids will have to be removed from the tank
[JI] may want to have a specially designed tank to enable easy removal of these solids
[JI] You may want to heat the solids to remove the vapour
[JI] assume two tanks because it will need cleaning out on a regular basis
[RM] assume that they would remove the residue by just opening a valve at the bottom of the tank without emptying the tank.
[JI] type of residue depend indicates whether we can just open a valve on the bottom of the tank
[JI] if you have heating coils, you have to design the tank to work at 100 degC
[RM] normally balanced for less chance of overfill
[RM] cleaning you can empty into other tank
[JI] if it does not split itself properly we could have problems
[RM] (to enable splitting) method of flow control.
[JI] method of flow control is very expensive - another 10 thousand pounds
[JI] if it all goes back into one tank the composition would be completely different in the two tanks
[RM] I'll put a system on here (when drawing the ELD) to allow for in-breathing and out-breathing when loading from storage
[JI] isolation valve for maintenance of the tank
[RM] someone could go over and shut it (an isolation valve)
[RM] pump this stuff in, you want to be able to disconnect your system safely
[RM] connector or some sealed couplings, when you have taken that off you don't suddenly want to get reverse flow of liquid and all going over the floor
[RM] when we put liquid in you don't want splash filling
[JI] you could put in a dip pipe with an anti-syphon hole (to avoid splash filling)
[RM] splash filling avoided by feeding in at the bottom
[JI] feeding in at bottom causes backflow
[RM] well we could get back flow from a syphon.
[JI] backflow from syphon depends on where your syphon hole is
[RM] if you put a dip pipe in you have to have an anti-syphon hole
[JI] splash filling you may get frothing
[JI] splash filling causes earthing problems
[JI] You have to put the bond across the joints and sometimes if you are using a non-

metallic system earthing electrodes or a ring to place in to earth the fluid itself

[RM] we have the choice of bottom filling, dip pipe and anti syphon for filling

[JI] bottom filling there is a risk of dumping 250 tonnes of stuff down to your filling point

[JI] may be we could put a refrigeration condenser above the tank to reduce the emissions

[JI] if you have not got any other major vents to an incinerator, the thermal oxidiser is a very expensive option

Appendix G. Facts determined from design case

Listed here are the facts and potential facts that were derived from the scenario. Potential facts are statements by the engineers to which they are uncertain, for example. *Toluene has a vapour pressure of 3 bar*. This fact was asserted in order progress the design, even though we did not have the right reference material available to tell us that this was fact. Potential facts have been listed to gain an understanding of what needs to be resolved. Facts that are not proven, or are questioned are marked in italics.

[RM] We are not connected up to a pipeline, therefore no continuous supply.
[RM] *Ground contaminated*
[RM] *Ground stable*
[JI] *Vessel need securing*
[JI] Store 200 tonnes of toluene (estimate)
[JI] *toluene is dry*
[JI] *toluene is pure*
[JI] *toluene is hot*
[JI] *toluene has a vapour pressure of 3 bar*
[JI] *nitrogen blanketting required*
[JI] *toluene polymerises in contact with air*
[JI] *blow gas directly through to incinerator*
[JI] *tank designed up to 18 inch water gauge*
[JI] *air and toluene vapour creates a flammable atmosphere*
[JI] *vent back to incinerator*
[JI] *vent to flare*
[JI] the normal design pressure for an atmospheric tank is 8 to 10 inch water gauge
[JI] *vessel design pressure is 12 inches water gauge*
[JI] pressure in tank will go up to 12 in a fire
[JI] toluene is unlikely to be allowed to vent to atmosphere
[RM] overflows can only work on atmospheric storage tanks
[RM] high pressure alarms required on tank
[JI] *vents can block*
[JI] *toluene can crystalise*
[JI] *tank is inherently safe*
[JI] *tank can take dead end pressure fead to it*
[JI] *tank can deal with fire relief*
[JI] *vent can block*
[RM] traces of substances in toluene
[RM] low melting point material
[RM] *freezing point of toluene is quite low*
[RM] *heating coil in tank*
[JI] recycle loop back to inlet vessel
[JI] *use floating roof tank*

[JI] *venting system required*
[JI] *toluene has a low volatility*
[JI] *toluene has a low toxicity*
[RM] large quantities of toluene are stored
[RM] our toluene requirement is 10 tonnes per hour
[RM] many tanker deliveries
[RM] large amount of road traffic
[RM] *tank down every two years to clean out sludge*
[RM] *fill one tank up and run one down*
[RM] *one tank out for inspection and maintenance*
[RM] *build one big tank*
[RM] *two tanks with one left empty*
[RM] *leak into empty tank*
[RM] CIMA ragulations stipulate maximum storage quantities
[RM] *cross contamination of products*
[RM] *fluid is not toluene*
[RM] *load tank with wrong material*
[JI] *incorrect tank connection*
[JI] *item of equipment is standard*
[RM] need a nice large tank, keep it topped up
[JI] we need to store 500m3, two days supply
[RM] *comply with SHE for storage*
[JI] purchase product at cheap time of year
[JI] store product off-site
[JI] *storage near process facility*
[JI] based at existing petrochemical works near plant
[RM] planning permission is needed
[RM] CIMA not considered
[RM] *toluene is flammable*
[RM] *toluene flashpoint is 10 degC*
[RM] *toluene is toxic*
[RM] *toluene has a residue*
[JI] *vessel contents can be contaminated*
[JI] benzene is present in the process
[JI] contamination is a function of re-cycle
[JI] toluene is flammable
[JI] nitrogen blanketting is required
[JI] flashpoint of toluene is 10 deg C
[JI] max operating temperature is below 10 deg C
[JI] vent to thermal oxidiser
[JI] not allowed to vent toluene to atmosphere
[JI] *have vents*
[RM] *use a floating roof vessel*
[JI] toluene has a high volatility

[JI] toluene has a high toxicity
[RM] tank has to breath
[RM] we use an atmospheric tank
[RM] thermal oxidiser is required
[RM] atmospheric operation
[RM] *use pressure vessel*
[RM] pressure vessels are expensive
[JI] *have a suction system*
[JI] *tank is sucked in*
[RM] an atmospheric tank will not withstand suction
[JI] *use back breakers*
[JI] *tank is under suction*
[JI] *system is under suction*
[JI] flammable vapour coming through
[JI] *vapour and air create flammable mixure*
[JI] *detonations and explosions in the vent header*
[JI] *thermal oxidiser is not working*
[JI] *have vent valves*
[JI] *vent valves are open*
[JI] *road tankers can discharge*
[RM] *pressure to oxidiser is .2 bar*
[JI] assume blown tankers at 2 bar gauge to unload
[RM] pump to unload
[RM] *use nitrogen pressure to unload*
[JI] *use air from tanker*
[JI] tankers do not have nitrogen
[RM] unload nitrogen can be provided by works
[JI] *tank is empty when unloading*
[JI] *pressure surge through pipe*
[RM] trip valve prevents pressure surge
[JI] trip valves may jam open
[JI] *fill from road tanker*
[RM] recycle from cracker to tank
[JI] you never crack everything, you always feed back
[JI] feedtank contaminated with rubbish
[RM] *contents of storage tank is pure*
[RM] *vent breather can block*
[JI] vent breather cannot block with toluene
[JI] *vents are kept free*
[RM] birds nest in vents
[RM] liquid is pumped from tanker
[RM] no gas breaktrough from tanker
[RM] there is a re-cycle from still
[JI] *loose level from still*

[JI] 6-8 m3/hour of vapour from still
[RM] minimum temperature is 0 deg C
[RM] dont need heating coils
[RM] re-cycle back from plant is hot
[JI] *solids are trapped*
[JI] solids will have to be removed from the tank
[JI] solids may be carsneogenic
[JI] solids may be toxic
[JI] *solids heated*
[JI] *remove vapour from solids*
[JI] two tanks required
[JI] tanks need cleaning on regular basis
[JI] two tanks cover availabilty
[JI] two tanks cover inspection
[JI] remove residue from tank
[JI] open valve at bottom of tank to remove residue
[JI] empty the tank
[JI] solids build up
[JI] we have to wash out solids
[JI] require heating coils in tank
[JI] design tank to work at 100 deg C
[RM] *One tank full*
[RM] *One tank empty*
[RM] *Both tanks full*
[RM] *Both tanks normally balanced*
[RM] *Recycle to both tanks*
[JI] Recycle not split properly
[RM] *Flow control needed*
[RM] Build tanks in mild steel
[RM] Allow for tank inbreathing
[RM] Allow for tank outbreathing
[RM] Tank loading from storage
[RM] *Tank sucked in*
[RM] Offload with a pump
[RM] Disconnect system
[RM] Reverse flow of liquid from tank
[RM] Liquid over the floor
[RM] Splash filling
[RM] Static buildup
[JI] *Dip pipe and anti-syphon hole*
[RM] *Feed in tank at bottom*
[JI] Back flow from tank
[JI] *frothing problem*
[JI] *earth bonding required*

[JI] *bond the joints*
[JI] *non-metalic system*
[JI] *earth the fluid*
[JI] *system corrosive*
[JI] *dump 250 tonnes of toluene down to filling point*
[JI] reduce emmisions
[JI] *vent to incinerator*
[JI] thermal oxidiser is expensive
[JI] *use refrigeration*

Appendix H. Actions in the design case

The following is a list of actions and requirements derived from the vessel design meeting. The actions and requirements have been presented in an order in which they are presented in the discussion. There may also be repetition.

[RM] tanker in product
[RM] need reliable continuous supply
[RM] need some means to continue production if deliveries can't be made
[RM] we needed enough inventory to keep the plant running for two days
[RM] product can be purchased and stored at a time of year when it is cheap to purchase
[RM] lets have a pressure rupture disk on the tank
[JI] cannot store product in an atmospheric pressure tank
[JI] determine type of flame traps, nitrogen blanketing etc.
[JI] nitrogen would go to a thermal oxidiser
[JI] back pressure problem
[JI] put fans in
[JI] use pressure vessel and blow vapour through to an incinerator
[JI] blow the gas directly through to the incinerators
[JI] put a suction system in
[JI] can draw in air
[JI] toluene mix with air and create a flammable atmosphere
[JI] create flammable headers
[JI] vent to a thermal oxidiser
[JI] we have nitrogen blanketing
[JI] vent back to an incinerator
[JI] never connect this to an incinerator
[JI] settle for an atmospheric tank
[JI] vent to a flare
[JI] vent to atmosphere.
[JI] high pressure alarms or reliefs on tank
[JI] design the tank to take the dead end pressure of what can be feed to it
[RM] heating coil to maintain a minimum temperature in the vessel
[JI] recycle loop back into the inlet vessel
[RM] tank should be nitrogen blanketed
[JI] we could have floating roof tanks
[RM] take it down every two years
[RM] clean out the sludge
[RM] two tanks minimum
[RM] fill one tank up and run one down
[RM] one tank out for inspection or maintenance
[RM] tank is leaking
[RM] spare tank to draw from
[RM] build one great big tank

[RM] You may have two tanks with one left empty so that if you have a leak you can leak into the empty tank
[JI] cross contamination
[JI] engineers change wrong connections
[RM] 10 tonnes an hour
[RM] we need road tankers queuing up
[RM] need a nice large tank

[RM] connecting the tanker pipes
[RM] Nitrogen blanketing
[JI] vent to a thermal oxidiser
[JI] vent toluene to atmosphere
[RM] have a floating roof vessel
[RM] need a vent
[RM] tank has to breath
[RM] need a thermal oxidiser
[RM] an atmospheric operation
[RM] put a pressure vessel in
[JI] we could have a suction system
[JI] sucking your tank
[JI] withstand suction
[JI] back-breakers, that blow
[JI] tank is under suction
[JI] system under suction
[JI] flammable vapour
[JI] make a flammable mixture
[JI] going towards the thermal oxidiser
[JI] flammable mixture
[JI] detonations and explosions in the vent header
[JI] You also have to allow for when the thermal oxidiser isn't working, or for some reason can't take stuff
[JI] vent to atmosphere or a flair stack
[JI] valve is open
[JI] suck in air
[JI] flammable atmosphere
[JI] vent to a thermal oxidiser
[JI] won't get the round tankers to discharge
[JI] filling
[JI] blown tankers (pressure on the top and blow it out)
[RM] empty tankers
[RM] pumping to empty tankers
[RM] The options to unload the tankers are to either pump it or use nitrogen pressure
[JI] air from a tanker
[RM] nitrogen was actually provided by the works

[JI] pressure with an open end through the pipe, up into the storage
[JI] pressure through the vent
[RM] prevent blow through with a trip valve
[JI] trip valve jam open
[RM] recycle to the storage tank
[JI] feed back from cracker
[JI] tank contaminated with a lot of rubbish
[RM] blockage of the vent breather
[JI] keep the vents free
[RM] birds nesting in vents
[JI] put the organics into the flame
[JI] kill all the halogens
[JI] flare stack provides a reasonable amount of combustion
[RM] liquid is pumped from the tanker
[RM] gas breakthrough from the road tanker
[RM] re-cycle from still
[RM] gas breakthrough to the tank possible
[JI] did loose level from still
[JI] gas break through to tank
[JI] 6-8 m cubed an hour of vapour from the still
[RM] hot re-cycle from plant
[JI] trap solids in tank
[JI] decomposition possible
[JI] solids will have to be removed from the tank
[JI] heat the solids
[JI] remove the vapour
[JI] cleaning out tank on regular basis
[JI] inspection tank
[JI] remove the residue from tank
[RM] opening a valve at the bottom of the tank
[RM] emptying the tank
[JI] heating coils
[JI] design the tank to work at 100 degC
[RM] want one full and the other empty
[RM] have both tanks full
[RM] both tanks normally balanced for less chance of overfill
[RM] cleaning the tanks
[RM] empty into other tank
[RM] re-cycle to each tank
[JI] split itself
[RM] method of flow control
[JI] re-cycle goes back into one tank
[JI] composition would be completely different in the two tanks
[RM] loss of nitrogen

[RM] loss of thermal oxidiser
[RM] reverse flow from benzene column
[RM] loss of nitrogen pressure control
[RM] in-breathing and out-breathing when loading from storage
[JI] maintenance of the tank
[RM] someone could go over and shut it
[JI] pressure control would still pick it up you have an indication if there is a problem
[RM] sucking the tank in
[RM] put a back breather on tank
[RM] we have got tanks coming in
[RM] off-load with a pump
[RM] able to disconnect system safely
[RM] reverse flow of liquid all going over the floor
[RM] want to avoid splash filling
[RM] feeding in at the bottom
[RM] back flow to feed
[RM] designing plants safely
[RM] splash filling
[RM] static buildup
[JI] frothing
[JI] splash filling
[JI] earthing
[JI] earth bonding
[JI] bond across the joints
[RM] bottom filling
[RM] dip pipe and anti syphon hole
[RM] bottom filling
[JI] put a refrigeration condenser above the tank
[JI] reduce the emissions
[JI] vents to an incinerator

Appendix I. Automated design of case scenario using CDEX

```
[TRANSFER transfer
 ..outlet-mass-flow-rate 10; unit-process-downstream MAIN::still; unit-process-upstream offsite;
  (synthesisedRefinement%gen5

   >>> synthProp%gen32, roger:  stream%gen33,
   > problem: conflict%gen86, roger, NO-PIPE  HARD
   >        Not connected to pipeline
   >            pref:0.10  max:0.25  rev:0.40  conf:1.00
   >            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10

   >>> synthProp%gen39, roger:  tanker%gen40, stream%gen46,
   > evaluation: gen87, roger: Lots of  traffic, low connection time
   >            pref:0.29  max:0.60  rev:0.48  conf:1
   >            pv:nil 0.60 0.36 nil nil  uv:0.25 0.30 0.30 0.05 0.10

   >>> synthProp%gen52, roger:  tanker%gen53, transfer%gen59, storage%gen65, transfer%gen71, recycle%gen77,
   > evaluation: gen88, roger: Buy stock when cheap
   >            pref:0.19  max:0.25  rev:0.76  conf:0.5
   >            pv:0.76 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > problem: conflict%gen91, jeremy, HIGH-STOCK MORE-FLOW  SOFT
   >        best not to hold stocksLoose level from still - 6-8m3 vapour
   >            pref:0.25  max:0.40  rev:0.63  conf:0.80
   >            pv:nil nil 0.63 nil nil  uv:0.20 0.30 0.40 0.00 0.10
   > accepted synthProp%gen52 <
   [TANKER tanker%gen53
    ..unit-process-downstream MAIN::transfer%gen59; unit-process-upstream offsite;
   ]
   [TRANSFER transfer%gen59
    ..unit-process-downstream MAIN::storage%gen65; unit-process-upstream MAIN::tanker%gen53;
     (synthesisedRefinement%gen63

>>> transFromTank%gen138, roger:  storage%gen151, stream%gen145, tripValve%gen139,
> evaluation: gen244, roger: Control valve prevents pressure blow through.
>            pref:0.26  max:0.30  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen250, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.30  max:0.40  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> transFromTank%gen157, roger:  stream%gen158, stream%gen164, pump%gen170,
> evaluation: gen245, roger: Pumping is apropriate
>            pref:0.00  max:0.00  rev:0.00  conf:0.8
>            pv:nil nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10

>>> transFromTank%gen190, jeremy:  gasStorage%gen191, stream%gen197,
> evaluation: gen247, roger: Control valve needed to prevent pressure blow through.
>            pref:0.16  max:0.30  rev:0.52  conf:0.8
>            pv:nil nil 0.52 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen253, jeremy: Pressure through transfer when tank empty
>            pref:0.34  max:0.40  rev:0.84  conf:1
>            pv:nil nil 0.84 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> gen292, roger:  gen293, gen299, gen305,
> evaluation: gen316, roger: Control valve prevents pressure blow through.
>            pref:0.29  max:0.33  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.23 0.30 0.33 0.03 0.10
> problem: conflict%gen317, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.25  max:0.33  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.23 0.30 0.33 0.03 0.10

>>> gen660, roger:  gen661, gen667, gen673,
> evaluation: gen804, roger: Control valve prevents pressure blow through.
```

```
>            pref:0.28  max:0.32  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.24 0.30 0.32 0.04 0.10
> problem: conflict%gen805, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.24  max:0.32  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.24 0.30 0.32 0.04 0.10

>>> gen1313, roger:  gen1314, gen1320, gen1326,
> evaluation: gen1492, roger: Control valve prevents pressure blow through.
>            pref:0.28  max:0.31  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.24 0.30 0.31 0.04 0.10
> problem: conflict%gen1493, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.24  max:0.31  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.24 0.30 0.31 0.04 0.10

>>> gen2350, roger:  gen2351, gen2357, gen2363,
> evaluation: gen2608, roger: Control valve prevents pressure blow through.
>            pref:0.27  max:0.31  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.31 0.05 0.10
> problem: conflict%gen2609, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.24  max:0.31  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.25 0.30 0.31 0.05 0.10

>>> gen3237, roger:  gen3238, gen3244, gen3250,
> evaluation: gen3381, roger: Control valve prevents pressure blow through.
>            pref:0.27  max:0.31  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.31 0.05 0.10
> problem: conflict%gen3382, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.23  max:0.31  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.25 0.30 0.31 0.05 0.10

>>> gen3878, roger:  gen3879, gen3885, gen3891,
> evaluation: gen4096, roger: Control valve prevents pressure blow through.
>            pref:0.27  max:0.30  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen4099, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.23  max:0.30  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.25 0.30 0.30 0.05 0.10

>>> gen6244, roger:  gen6245, gen6251, gen6257,
> evaluation: gen6278, roger: Control valve prevents pressure blow through.
>            pref:0.27  max:0.30  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen6279, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.23  max:0.30  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.25 0.30 0.30 0.05 0.10

>>> gen7324, roger:  gen7325, gen7331, gen7337,
> evaluation: gen7344, roger: Control valve prevents pressure blow through.
>            pref:0.27  max:0.30  rev:0.88  conf:0.8
>            pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen7345, jeremy, MORE_FLOW  SOFT
>        Trip valve may jam open
>            pref:0.23  max:0.30  rev:0.76  conf:0.80
>            pv:nil nil 0.76 nil nil  uv:0.25 0.30 0.30 0.05 0.10
<> CR strategy:    (compromise%gen259, MAIN::transFromTank%gen138)
                   (MAIN::gen292 as alternative to MAIN::transFromTank%gen138)
                   (consensus%gen264, MAIN::transFromTank%gen138)
                   (compromise%gen259, MAIN::gen292)
                   (MAIN::gen660 as alternative to MAIN::gen292)
                   (consensus%gen264, MAIN::gen660)
                   (compromise%gen259, MAIN::gen660)
```

```
                (MAIN::gen1313 as alternative to MAIN::gen660)
                (consensus%gen264, MAIN::gen1313)
                (compromise%gen259, MAIN::gen1313)
                (MAIN::gen2350 as alternative to MAIN::gen1313)
                (consensus%gen264, MAIN::gen2350)
                (compromise%gen259, MAIN::gen2350)
                (MAIN::gen3237 as alternative to MAIN::gen2350)
                (consensus%gen264, MAIN::gen3237)
                (compromise%gen259, MAIN::gen3237)
                (MAIN::gen3878 as alternative to MAIN::gen3237)
                (consensus%gen264, MAIN::gen3878)
                (majorityRule%gen265, MAIN::transFromTank%gen138)
                (majorityRule%gen265, MAIN::gen2350)
                (majorityRule%gen265, MAIN::gen3237)
                (majorityRule%gen265, MAIN::gen3878)
                (compromise%gen259, MAIN::gen3878)
                (MAIN::gen6244 as alternative to MAIN::gen3878)
                (consensus%gen264, MAIN::gen6244)
                (majorityRule%gen265, MAIN::gen6244)
                (majorityRule%gen265, MAIN::gen292)
                (majorityRule%gen265, MAIN::gen660)
                (compromise%gen259, MAIN::gen6244)
                (MAIN::gen7324 as alternative to MAIN::gen6244)
                (consensus%gen264, MAIN::gen7324)
                (majorityRule%gen265, MAIN::gen7324) <>
> accepted gen7324 <
[STORAGE gen7325
 ..location offsite; fluid nitrogen; reliability 100; unit-process-downstream MAIN::tanker%gen53;
  (parametisedRefinement%gen7326

   >>> nitroParamProp%gen7358, roger:  storage%gen7359,
   > evaluation: gen7454, roger: Nitrogen is available, prefer works
   >            pref:0.24  max:0.30  rev:0.80  conf:1
   >            pv:nil 0.80 nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > problem: conflict%gen7459, jeremy, NOT_POSSIBLE  HARD
   >       Nitrogen supply not available on tanker
   >            pref:0.15  max:0.30  rev:0.50  conf:0.50
   >            pv:nil 0.50 nil nil nil  uv:0.20 0.30 0.40 0.00 0.10

   >>> nitroParamProp%gen7365, roger:  storage%gen7366,
   > evaluation: gen7458, roger: Nitrogen is available, prefer works
   >            pref:0.27  max:0.30  rev:0.90  conf:1
   >            pv:nil 0.90 nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
   <> CR strategy:  (consensus%gen7481, MAIN::nitroParamProp%gen7358) <>
   > accepted nitroParamProp%gen7358 <
   [STORAGE storage%gen7359
   ]
  )
  (synthesisedRefinement%gen7329

   >>> heatedTank%gen7503, jeremy:  storage%gen7504, coil%gen7511,
   > evaluation: gen7530, jeremy: Good for maintenace
   >            pref:0.10  max:0.10  rev:1.00  conf:1
   >            pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
   > problem: conflict%gen7528, roger, NOT-REQUIRED  HARD
   >       Heating coil not required
   >            pref:0.10  max:0.25  rev:0.40  conf:1.00
   >            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > Proposal was NOT accepted
  )
]
[STREAM gen7331
 ..unit-process-downstream MAIN::storage%gen65; unit-process-upstream MAIN::tripValve%gen139;
  (synthesisedRefinement%gen7335

   >>> streamToTank%gen7374, roger:  stream%gen7375, stream%gen7381, isolValve%gen7387,
   > evaluation: gen7466, roger: Connector flow in environment;Reverse flow from connector;
```

```
>                    Potential for splash filling;Control valve shut;
>           pref:0.11  max:0.30  rev:0.36  conf:0.7
>           pv:nil nil 0.36 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen7469, jeremy: Iso. valve ok for maint, frothing potential
>           pref:0.48  max:0.80  rev:0.60  conf:1
>           pv:nil 0.20 0.80 nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

>>> streamToTank%gen7393, roger:  stream%gen7394, stream%gen7400, isolValve%gen7406,
                                  connector%gen7412,
> evaluation: gen7467, roger: Potential for splash filling;Control valve shut;
>           pref:0.16  max:0.30  rev:0.52  conf:0.8
>           pv:nil nil 0.52 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen7471, jeremy: Iso. valve ok for maint, frothing potential
>           pref:0.48  max:0.80  rev:0.60  conf:1
>           pv:nil 0.20 0.80 nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

>>> transFromTank%gen7418, jeremy:  stream%gen7419, stream%gen7425, isolValve%gen7431,
                          connector%gen7437, dipPipe%gen7443,
> evaluation: gen7468, roger: Control valve shut;
>           pref:0.26  max:0.30  rev:0.88  conf:0.8
>           pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen7473, jeremy: Iso. Valve good for maint, dip pipe prevents static
>           pref:0.50  max:0.50  rev:1.00  conf:1
>           pv:nil nil 1.00 nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
> accepted transFromTank%gen7418 <
[STREAM stream%gen7419
 ..unit-process-downstream MAIN::isolValve%gen7431; unit-process-upstream MAIN::tripValve%gen139;
]
[STREAM stream%gen7425
 ..unit-process-downstream MAIN::connector%gen7437; unit-process-upstream MAIN::isolValve%gen7431;
]
[ISOLATION_VALVE isolValve%gen7431
 ..unit-process-downstream MAIN::stream%gen7425; unit-process-upstream MAIN::stream%gen7419;
]
[CONNECTOR connector%gen7437
  ..unit-process-downstream MAIN::dipPipe%gen7443; unit-process-upstream MAIN::stream%gen7425;
  ]
  [DIP_PIPE dipPipe%gen7443
   ..unit-process-downstream MAIN::storage%gen65; unit-process-upstream MAIN::connector%gen7437;
  ]
 )
]
[CONTROL_VALVE gen7337
 ..unit-process-downstream MAIN::stream%gen145; unit-process-upstream MAIN::tanker%gen53;
]
)
]
[STORAGE storage%gen65
 ..reliability 90; unit-process-downstream MAIN::transfer%gen71; unit-process-upstream MAIN::transfer%gen59;
 (synthesisedRefinement%gen69

  >>> storageFarm%gen177, roger:  storage%gen178, storage%gen184,
  > evaluation: gen246, roger: >1 tank good for maintenance
  >           pref:0.10  max:0.10  rev:1.00  conf:1
  >           pv:nil nil nil nil 1.00  uv:0.25 0.30 0.30 0.05 0.10
  > evaluation: gen252, jeremy: Re-cycle without flow control problems, control expensive, >1 tank ok for maintenance
  >           pref:0.33  max:0.60  rev:0.55  conf:1
  >           pv:0.40 0.50 nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

  >>> heatedTank%gen203, jeremy:  storage%gen204, coil%gen211,
  > evaluation: gen255, jeremy: Poor for maintenance
  >           pref:0.02  max:0.10  rev:0.20  conf:1
  >           pv:nil nil nil nil 0.20  uv:0.20 0.30 0.40 0.00 0.10
  > problem: conflict%gen248, roger, NOT-REQUIRED  HARD
  >       Heating coil not required
  >           pref:0.10  max:0.25  rev:0.40  conf:1.00
  >           pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
```

```
>>> storageFarm%gen217, jeremy:  storage%gen218, storage%gen224, pipe%gen230,
> evaluation: gen249, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen257, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.35  max:0.60  rev:0.58  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

>>> gen965, jeremy:  gen966, gen972, gen978,
> evaluation: gen1117, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.22 0.30 0.37 0.02 0.10
> evaluation: gen1135, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.36  max:0.62  rev:0.59  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.22 0.30 0.37 0.02 0.10

>>> gen2579, jeremy:  gen2580, gen2586, gen2592,
> evaluation: gen2620, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.21 0.30 0.38 0.01 0.10
> evaluation: gen2626, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.36  max:0.61  rev:0.59  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.21 0.30 0.38 0.01 0.10

>>> gen3139, jeremy:  gen3140, gen3146, gen3152,
> evaluation: gen3378, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.21 0.30 0.39 0.01 0.10
> evaluation: gen3380, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.36  max:0.61  rev:0.59  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.21 0.30 0.39 0.01 0.10

>>> gen3462, jeremy:  gen3463, gen3469, gen3475,
> evaluation: gen3570, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10
> evaluation: gen3616, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.35  max:0.60  rev:0.59  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10

>>> gen4006, jeremy:  gen4007, gen4013, gen4019,
> evaluation: gen4304, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10
> evaluation: gen4306, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.35  max:0.60  rev:0.58  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10

>>> gen6125, jeremy:  gen6126, gen6132, gen6138,
> evaluation: gen6168, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
> evaluation: gen6172, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.35  max:0.60  rev:0.58  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
<> CR strategy:    (smoothing%gen279, MAIN::storageFarm%gen217)
                   (smoothing%gen279, MAIN::storageFarm%gen177)
                   (compromise%gen275, MAIN::storageFarm%gen217)
                   (MAIN::gen965 as alternative to MAIN::storageFarm%gen217)
                   (smoothing%gen279, MAIN::gen965)
                   (consensus%gen280, MAIN::gen965)
                   (compromise%gen275, MAIN::gen965)
                   (MAIN::gen2579 as alternative to MAIN::gen965)
                   (consensus%gen280, MAIN::gen2579)
                   (compromise%gen275, MAIN::gen2579)
                   (MAIN::gen3139 as alternative to MAIN::gen2579)
                   (consensus%gen280, MAIN::gen3139)
```

```
                (compromise%gen275, MAIN::gen3139)
                (MAIN::gen3462 as alternative to MAIN::gen3139)
                (consensus%gen280, MAIN::gen3462)
                (smoothing%gen279, MAIN::gen2579)
                (compromise%gen275, MAIN::gen3462)
                (MAIN::gen4006 as alternative to MAIN::gen3462)
                (consensus%gen280, MAIN::gen4006)
                (smoothing%gen279, MAIN::gen3139)
                (majorityRule%gen281, MAIN::gen965)
                (majorityRule%gen281, MAIN::gen2579)
                (compromise%gen275, MAIN::gen4006)
                (MAIN::gen6125 as alternative to MAIN::gen4006)
                (consensus%gen280, MAIN::gen6125)
                (majorityRule%gen281, MAIN::gen3139)
                (majorityRule%gen281, MAIN::gen3462)
                (majorityRule%gen281, MAIN::gen4006) <>
> accepted gen4006 <
[STORAGE gen4007
 ..material-of-construction mildsteel; reliability 99; unit-process-downstream MAIN::transfer%gen71;
                                                        unit-process-upstream MAIN::transfer%gen59;
  (synthesisedRefinement%gen4011

    >>> heatedTank%gen4756, jeremy:  storage%gen4757, coil%gen4763,
    > evaluation: gen5061, jeremy: Good for maintenace
    >           pref:0.10  max:0.10  rev:1.00  conf:1
    >           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
    > problem: conflict%gen5051, roger, NOT-REQUIRED  HARD
    >       Heating coil not required
    >           pref:0.10  max:0.25  rev:0.40  conf:1.00
    >           pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
    > Proposal was NOT accepted
  )
  (selectedRefinement%gen4009

    >>> storageSelProp%gen4735, jeremy:  atmosTank%gen4736,
    > evaluation: gen5048, roger: Not as expensive as pressure vessel
    >           pref:0.25  max:0.25  rev:1.00  conf:1
    >           pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
    > problem: conflict%gen5058, jeremy, NOT_POSSIBLE  HARD
    >       Cannot store hot toluene in atmos. tank
    >           pref:0.35  max:0.40  rev:0.88  conf:0.30
    >           pv:nil nil 0.88 nil nil  uv:0.20 0.30 0.40 0.00 0.10

    >>> storageSelProp%gen4742, jeremy:  floatingRoofTank%gen4743,
    > evaluation: gen5049, roger: Not as expensive as pressure vessel
    >           pref:0.25  max:0.25  rev:1.00  conf:1
    >           pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
    > problem: conflict%gen5059, jeremy, NOT_POSSIBLE  HARD
    >       Highly volatile and toxic fluid, floating roof not appropriate
    >           pref:0.00  max:0.00  rev:0.00  conf:1.00
    >           pv:nil nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10

    >>> storageSelProp%gen4749, jeremy:  pressureVessel%gen4750,
    > evaluation: gen5050, roger: Pressure vessel expensive
    >           pref:0.10  max:0.25  rev:0.40  conf:1
    >           pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
    > evaluation: gen5060, jeremy: Costly system to get tank to discharge to pressure vessel
    >           pref:0.00  max:0.20  rev:0.00  conf:1
    >           pv:0.00 nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10
    > accepted storageSelProp%gen4735 <
    [ATMOSPHERIC_TANK atmosTank%gen4736
     (synthesisedRefinement%gen4740

       >>> atmosTankSynth%gen5512, jeremy:  atmosTank%gen5513, blanket%gen5519, suctionSystem%gen5525,
                                                                                    valve%gen5531,
       > evaluation: gen5654, jeremy: Valve good for removing solids; Good for maintenace
       >           pref:0.10  max:0.10  rev:1.00  conf:1
```

```
>            pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
> accepted atmosTankSynth%gen5512 <
[ATMOSPHERIC_TANK atmosTank%gen5513
 ..material-of-construction mildsteel;
  (parametisedRefinement%gen5514

    >>> paramProp%gen5744, roger:  storage%gen5746,
    > accepted paramProp%gen5744 <
    [ATMOSPHERIC_TANK storage%gen5746
    ]
  )
]
[BLANKET blanket%gen5519
 ..unit-process-downstream MAIN::atmosTank%gen5513; unit-process-upstream MAIN::atmosTank%gen5513;
  (synthesisedRefinement%gen5523

    >>> blanket%gen5784, roger:  fan%gen5785, stream%gen5791, oxidiser%gen5797, trip%gen5803,
    > evaluation: gen5985, jeremy: Expensive;No vent with TO;Back pressure problem;
    >            pref:0.33  max:0.60  rev:0.55  conf:0.6
    >            pv:0.30 nil 0.67 nil nil  uv:0.20 0.30 0.40 0.00 0.10

    >>> blanket%gen5822, jeremy:  fan%gen5823, stream%gen5829, oxidiser%gen5835,
    > evaluation: gen5988, jeremy: Expensive;No vent with TO;Back pressure problem;
    >            pref:0.33  max:0.60  rev:0.55  conf:0.6
    >            pv:0.30 nil 0.67 nil nil  uv:0.20 0.30 0.40 0.00 0.10

    >>> blanket%gen5841, jeremy:  fan%gen5842, stream%gen5848, oxidiser%gen5854, vent%gen5860,
                                              trip%gen5866,
    > evaluation: gen5981, roger: Birds can next in the vents.
    >            pref:0.21  max:0.30  rev:0.70  conf:0.3
    >            pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > evaluation: gen5990, jeremy: Expensive;Backpressure with TO resolved with vent;Vent open creates flam.
                                    atmos.;
                          Back pressure problem;Vent can block;
    >            pref:0.39  max:0.60  rev:0.65  conf:0.6
    >            pv:0.30 nil 0.82 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> blanket%gen5872, jeremy:  vent%gen5873, pipe%gen5879, flare%gen5885,
> evaluation: gen5982, roger: Birds can next in the vents.
>            pref:0.21  max:0.30  rev:0.70  conf:0.3
>            pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen5992, jeremy: Cheaper;Flair potentially poor solution for future;Vent can block;
>            pref:0.52  max:0.60  rev:0.87  conf:0.5
>            pv:0.80 nil 0.90 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> blanket%gen5891, jeremy:  vent%gen5892, pipe%gen5898, condenser%gen5904,
> evaluation: gen5983, roger: Birds can next in the vents.
>            pref:0.21  max:0.30  rev:0.70  conf:0.3
>            pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen5994, jeremy: Cheaper;Vent can block;
>            pref:0.52  max:0.60  rev:0.87  conf:0.6
>            pv:0.80 nil 0.90 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> blanket%gen5910, jeremy:  vent%gen5911, pipe%gen5917, atmosphere%gen5923,
> evaluation: gen5984, roger: Birds can next in the vents.
>            pref:0.21  max:0.30  rev:0.70  conf:0.3
>            pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen5996, jeremy, NOT_POSSIBLE  HARD
>        Not allowed to vent toluene to atmosphere.
>            pref:0.00  max:0.00  rev:0.00  conf:1.00
>            pv: uv:

>>> gen6106, jeremy:  gen6107, gen6113, gen6119,
> evaluation: gen6167, roger: Birds can next in the vents.
>            pref:0.26  max:0.37  rev:0.70  conf:0.3
>            pv:nil nil 0.70 nil nil  uv:0.22 0.30 0.37 0.02 0.10
> evaluation: gen6169, jeremy: Cheaper;Vent can block;
```

```
  >            pref:0.50  max:0.58  rev:0.86  conf:0.6
  >            pv:0.80 nil 0.90 nil nil  uv:0.22 0.30 0.37 0.02 0.10
  <> CR strategy:    (compromise%gen6055, MAIN::blanket%gen5891)
                      (MAIN::gen6106 as alternative to MAIN::blanket%gen5891)
                      (consensus%gen6060, MAIN::blanket%gen5891) <>
  > accepted blanket%gen5891 <
  [VENT vent%gen5892
   ..unit-process-downstream MAIN::pipe%gen5898; unit-process-upstream MAIN::atmosTank%gen5513;
  ]
  [PIPE pipe%gen5898
   ..unit-process-downstream MAIN::condenser%gen5904; unit-process-upstream MAIN::vent%gen5892;
  ]
  [REFRIGERATION_CONDENSER condenser%gen5904
   ..unit-process-upstream MAIN::pipe%gen5917;
  ]
 )
]
[SUCTION_SYSTEM suctionSystem%gen5525
..unit-process-downstream MAIN::atmosTank%gen5513; unit-process-upstream MAIN::atmosTank%gen5513;
 (synthesisedRefinement%gen5529

  >>> suctionProp%gen5809, roger:  storage%gen5810, stream%gen5816,
  > problem: conflict%gen5987, jeremy, HIGH-STOCK  SOFT
  >       best not to hold stocks
  >            pref:0.39  max:0.40  rev:0.96  conf:0.80
  >            pv:nil nil 0.96 nil nil  uv:0.20 0.30 0.40 0.00 0.10

  >>> suctionSysProp%gen5929, jeremy:  suctionSystem%gen5930,
  > evaluation: gen5998, jeremy: Back breaker blow and create flammable atmosphere
  >            pref:0.25  max:0.40  rev:0.62  conf:1
  >            pv:nil nil 0.62 nil nil  uv:0.20 0.30 0.40 0.00 0.10
  > accepted suctionProp%gen5809 <
  [STORAGE storage%gen5810
   ..fluid nitrogen; reliability 100; unit-process-downstream MAIN::stream%gen5816;
    (synthesisedRefinement%gen5814

     >>> heatedTank%gen6175, jeremy:  storage%gen6176, coil%gen6183,
     > evaluation: gen6272, jeremy: Good for maintenace
     >            pref:0.10  max:0.10  rev:1.00  conf:1
     >            pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
     > problem: conflict%gen6270, roger, NOT-REQUIRED  HARD
     >       Heating coil not required
     >            pref:0.10  max:0.25  rev:0.40  conf:1.00
     >            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
     > Proposal was NOT accepted
    )
   ]
   [STREAM stream%gen5816
    ..unit-process-downstream MAIN::atmosTank%gen5513; unit-process-upstream MAIN::storage%gen5810;
   ]
  )
 ]
 [VALVE valve%gen5531
  ..unit-process-downstream MAIN::atmosTank%gen5513;
 ]
)
]
[STORAGE gen4013
 ..material-of-construction mildsteel; reliability 99; unit-process-upstream MAIN::pipe%gen230;
  (synthesisedRefinement%gen4017

   >>> heatedTank%gen4790, jeremy:  storage%gen4791, coil%gen4797,
   > evaluation: gen5065, jeremy: Good for maintenace
   >            pref:0.10  max:0.10  rev:1.00  conf:1
   >            pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
```

```
 > problem: conflict%gen5055, roger, NOT-REQUIRED  HARD
 >        Heating coil not required
 >            pref:0.10  max:0.25  rev:0.40  conf:1.00
 >            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > Proposal was NOT accepted
)
(selectedRefinement%gen4015

 >>> storageSelProp%gen4769, jeremy:  atmosTank%gen4770,
 > evaluation: gen5052, roger: Not as expensive as pressure vessel
 >            pref:0.25  max:0.25  rev:1.00  conf:1
 >            pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > problem: conflict%gen5062, jeremy, NOT_POSSIBLE  HARD
 >        Cannot store hot toluene in atmos. tank
 >            pref:0.35  max:0.40  rev:0.88  conf:0.30
 >            pv:nil nil 0.88 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> storageSelProp%gen4776, jeremy:  floatingRoofTank%gen4777,
 > evaluation: gen5053, roger: Not as expensive as pressure vessel
 >            pref:0.25  max:0.25  rev:1.00  conf:1
 >            pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > problem: conflict%gen5063, jeremy, NOT_POSSIBLE  HARD
 >        Highly volatile and toxic fluid, floating roof not appropriate
 >            pref:0.00  max:0.00  rev:0.00  conf:1.00
 >            pv:nil nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> storageSelProp%gen4783, jeremy:  pressureVessel%gen4784,
 > evaluation: gen5054, roger: Pressure vessel expensive
 >            pref:0.10  max:0.25  rev:0.40  conf:1
 >            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > evaluation: gen5064, jeremy: Costly system to get tank to discharge to pressure vessel
 >            pref:0.00  max:0.20  rev:0.00  conf:1
 >            pv:0.00 nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10
 > accepted storageSelProp%gen4769 <
 [ATMOSPHERIC_TANK atmosTank%gen4770
   ]
  )
 ]
 [PIPE gen4019
  ..unit-process-downstream MAIN::storage%gen224; unit-process-upstream MAIN::storage%gen218;
 ]
)
]
[TRANSFER transfer%gen71
 ..unit-process-downstream MAIN::still; unit-process-upstream MAIN::storage%gen65;
]
[TRANSFER recycle%gen77
 ..fluid-contaminants particulates; unit-process-downstream MAIN::storage%gen65; unit-process-upstream MAIN::still;
  (parametisedRefinement%gen78

   >>> transParamProp%gen113, jeremy:  transfer%gen114,
   > accepted transParamProp%gen113 <
   [TRANSFER transfer%gen114
   ]
  )
 ]
)
]
```

Appendix J. Case scenario design by quickest route

Appendix I shows the development of a detailed design for the given requirement (transfer to still). This appendix is given the same requirement with the exception that the framework has been instructed to find a solution in a quicker time frame.

The key to the hierachy is outlines in appenix I. Note that all accepted proposals are in bold.

```
[TRANSFER transfer
 ..outlet-mass-flow-rate 10; unit-process-downstream MAIN::still; unit-process-upstream offsite;
 (synthesisedRefinement%gen5

   >>> synthProp%gen32, roger:  stream%gen33,
   > problem: conflict%gen86, roger, NO-PIPE  HARD
   >        Not connected to pipeline
   >            pref:0.10  max:0.25  rev:0.40  conf:1.00
   >            pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10

   >>> synthProp%gen39, roger:  tanker%gen40, stream%gen46,
   > evaluation: gen87, roger: Lots of  traffic, low connection time
   >            pref:0.29  max:0.60  rev:0.48  conf:1
   >            pv:nil 0.60 0.36 nil nil  uv:0.25 0.30 0.30 0.05 0.10

   >>> synthProp%gen52, roger:  tanker%gen53, transfer%gen59, storage%gen65, transfer%gen71,
                                                          recycle%gen77,
   > evaluation: gen88, roger: Buy stock when cheap
   >            pref:0.19  max:0.25  rev:0.76  conf:0.5
   >            pv:0.76 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > problem: conflict%gen91, jeremy, HIGH-STOCK MORE-FLOW  SOFT
   >        best not to hold stocksLoose level from still - 6-8m3 vapour
   >            pref:0.25  max:0.40  rev:0.63  conf:0.80
   >            pv:nil nil 0.63 nil nil  uv:0.20 0.30 0.40 0.00 0.10
   > accepted synthProp%gen52 <
[TANKER tanker%gen53
 ..unit-process-downstream MAIN::transfer%gen59; unit-process-upstream offsite;
]
[TRANSFER transfer%gen59
 ..unit-process-downstream MAIN::storage%gen65; unit-process-upstream MAIN::tanker%gen53;
  (synthesisedRefinement%gen63

   >>> transFromTank%gen138, roger:  storage%gen151, stream%gen145, tripValve%gen139,
   > evaluation: gen244, roger: Control valve prevents pressure blow through.
   >            pref:0.26  max:0.30  rev:0.88  conf:0.8
   >            pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > problem: conflict%gen250, jeremy, MORE_FLOW  SOFT
   >        Trip valve may jam open
   >            pref:0.30  max:0.40  rev:0.76  conf:0.80
   >            pv:nil nil 0.76 nil nil  uv:0.20 0.30 0.40 0.00 0.10

   >>> transFromTank%gen157, roger:  stream%gen158, stream%gen164, pump%gen170,
   > evaluation: gen245, roger: Pumping is apropriate
   >            pref:0.00  max:0.00  rev:0.00  conf:0.8
   >            pv:nil nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10

   >>> transFromTank%gen190, jeremy:  gasStorage%gen191, stream%gen197,
   > evaluation: gen247, roger: Control valve needed to prevent pressure blow through.
   >            pref:0.16  max:0.30  rev:0.52  conf:0.8
   >            pv:nil nil 0.52 nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > evaluation: gen253, jeremy: Pressure through transfer when tank empty
   >            pref:0.34  max:0.40  rev:0.84  conf:1
```

```
>          pv:nil nil 0.84 nil nil  uv:0.20 0.30 0.40 0.00 0.10

>>> transFromTank%gen292, jeremy:  gasStorage%gen293, stream%gen299,
> evaluation: gen326, roger: Control valve needed to prevent pressure blow through.
>          pref:0.19  max:0.37  rev:0.52  conf:0.8
>          pv:nil nil 0.52 nil nil  uv:0.22 0.30 0.37 0.02 0.10
> evaluation: gen328, jeremy: Pressure through transfer when tank empty
>          pref:0.31  max:0.37  rev:0.84  conf:1
>          pv:nil nil 0.84 nil nil  uv:0.22 0.30 0.37 0.02 0.10
<> CR strategy:  (compromise%gen259, MAIN::transFromTank%gen190)
                 (MAIN::transFromTank%gen292 as alternative to MAIN::transFromTank%gen190) <>
> accepted transFromTank%gen138 <
[STORAGE storage%gen151
 ..location offsite; fluid nitrogen; reliability 100; unit-process-downstream MAIN::tanker%gen53;
  (parametisedRefinement%gen152

   >>> nitroParamProp%gen351, roger:  storage%gen352,
   > evaluation: gen458, roger: Nitrogen is available, prefer works
   >          pref:0.24  max:0.30  rev:0.80  conf:1
   >          pv:nil 0.80 nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > problem: conflict%gen466, jeremy, NOT_POSSIBLE  HARD
   >       Nitrogen supply not available on tanker
   >          pref:0.15  max:0.30  rev:0.50  conf:0.50
   >          pv:nil 0.50 nil nil nil  uv:0.20 0.30 0.40 0.00 0.10

   >>> nitroParamProp%gen358, roger:  storage%gen359,
   > evaluation: gen462, roger: Nitrogen is available, prefer works
   >          pref:0.27  max:0.30  rev:0.90  conf:1
   >          pv:nil 0.90 nil nil nil  uv:0.25 0.30 0.30 0.05 0.10

   >>> airParamProp%gen365, jeremy:  storage%gen366,
   <> CR strategy:  (consensus%gen570, MAIN::nitroParamProp%gen351) <>
   > accepted nitroParamProp%gen351 <
   [STORAGE storage%gen352
   ]
  )
  (synthesisedRefinement%gen155

   >>> heatedTank%gen619, jeremy:  storage%gen620, coil%gen627,
   > evaluation: gen722, jeremy: Good for maintenace
   >          pref:0.10  max:0.10  rev:1.00  conf:1
   >          pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
   > problem: conflict%gen706, roger, NOT-REQUIRED  HARD
   >       Heating coil not required
   >          pref:0.10  max:0.25  rev:0.40  conf:1.00
   >          pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > Proposal was NOT accepted
  )
]
[STREAM stream%gen145
 ..unit-process-downstream MAIN::storage%gen65; unit-process-upstream MAIN::tripValve%gen139;
  (synthesisedRefinement%gen149

   >>> streamToTank%gen375, roger:  stream%gen376, stream%gen382, isolValve%gen388,
   > evaluation: gen547, roger: Connector flow in environment;Reverse flow from connector;
                                                  Potential for splash filling;Control valve shut;
   >          pref:0.11  max:0.30  rev:0.36  conf:0.7
   >          pv:nil nil 0.36 nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > evaluation: gen550, jeremy: Iso. valve ok for maint, frothing potential
   >          pref:0.48  max:0.80  rev:0.60  conf:1
   >          pv:nil 0.20 0.80 nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

   >>> streamToTank%gen394, roger:  stream%gen395, stream%gen401, isolValve%gen407,
                                                          connector%gen413,
   > evaluation: gen548, roger: Potential for splash filling;Control valve shut;
   >          pref:0.16  max:0.30  rev:0.52  conf:0.8
   >          pv:nil nil 0.52 nil nil  uv:0.25 0.30 0.30 0.05 0.10
```

```
> evaluation: gen552, jeremy: Iso. valve ok for maint, frothing potential
>          pref:0.48  max:0.80  rev:0.60  conf:1
>          pv:nil 0.20 0.80 nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

>>> transFromTank%gen419, jeremy:  stream%gen420, stream%gen426, isolValve%gen432,
                                                    connector%gen438, dipPipe%gen444,
> evaluation: gen549, roger: Control valve shut;
>          pref:0.26  max:0.30  rev:0.88  conf:0.8
>          pv:nil nil 0.88 nil nil  uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen554, jeremy: Iso. Valve good for maint, dip pipe prevents static
>          pref:0.50  max:0.50  rev:1.00  conf:1
>          pv:nil nil 1.00 nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
> accepted transFromTank%gen419 <
[STREAM stream%gen420
 ..unit-process-downstream MAIN::isolValve%gen432; unit-process-upstream
            MAIN::tripValve%gen139;
]
[STREAM stream%gen426
 ..unit-process-downstream MAIN::connector%gen438; unit-process-upstream
            MAIN::isolValve%gen432;
]
[ISOLATION_VALVE isolValve%gen432
   ..unit-process-downstream MAIN::stream%gen426; unit-process-upstream
            MAIN::stream%gen420;
]
[CONNECTOR connector%gen438
 ..unit-process-downstream MAIN::dipPipe%gen444; unit-process-upstream
   MAIN::stream%gen426;
]
[DIP_PIPE dipPipe%gen444
 ..unit-process-downstream MAIN::storage%gen65;
   unit-process-upstream MAIN::connector%gen438;
]
 )
]
  [CONTROL_VALVE tripValve%gen139
   ..unit-process-downstream MAIN::stream%gen145; unit-process-upstream MAIN::tanker%gen53;
  ]
 )
]
[STORAGE storage%gen65
 ..reliability 90; unit-process-downstream MAIN::transfer%gen71; unit-process-upstream
        MAIN::transfer%gen59;
 (synthesisedRefinement%gen69

  >>> storageFarm%gen177, roger:  storage%gen178, storage%gen184,
  > evaluation: gen246, roger: >1 tank good for maintenance
  >          pref:0.10  max:0.10  rev:1.00  conf:1
  >          pv:nil nil nil nil 1.00  uv:0.25 0.30 0.30 0.05 0.10
  > evaluation: gen252, jeremy: Re-cycle without flow control problems, control expensive, >1 tank ok
                                                    for maintenance
  >          pref:0.33  max:0.60  rev:0.55  conf:1
  >          pv:0.40 0.50 nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

  >>> heatedTank%gen203, jeremy:  storage%gen204, coil%gen211,
  > evaluation: gen255, jeremy: Poor for maintenance
  >          pref:0.02  max:0.10  rev:0.20  conf:1
  >          pv:nil nil nil nil 0.20  uv:0.20 0.30 0.40 0.00 0.10
  > problem: conflict%gen248, roger, NOT-REQUIRED  HARD
  >       Heating coil not required
  >          pref:0.10  max:0.25  rev:0.40  conf:1.00
  >          pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10

  >>> storageFarm%gen217, jeremy:  storage%gen218, storage%gen224, pipe%gen230,
  > evaluation: gen249, roger: >1 tank good for maintenance
  >          pref:0.10  max:0.10  rev:1.00  conf:1
  >          pv:nil nil nil nil 1.00  uv:0.25 0.30 0.30 0.05 0.10
```

```
> evaluation: gen257, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.35  max:0.60  rev:0.58  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10

>>> gen305, jeremy:  gen306, gen312, gen318,
> evaluation: gen327, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.22 0.30 0.37 0.02 0.10
> evaluation: gen330, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.36  max:0.62  rev:0.59  conf:1
>           pv:0.80 0.30 nil nil 1.00  uqv:0.22 0.30 0.37 0.02 0.10

>>> gen966, jeremy:  gen967, gen973, gen979,
> evaluation: gen1017, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.21 0.30 0.38 0.01 0.10
> evaluation: gen1029, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.36  max:0.61  rev:0.59  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.21 0.30 0.38 0.01 0.10

>>> gen2688, jeremy:  gen2689, gen2695, gen2701,
> evaluation: gen2805, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.21 0.30 0.39 0.01 0.10
> evaluation: gen2810, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.36  max:0.61  rev:0.59  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.21 0.30 0.39 0.01 0.10

>>> gen3683, jeremy:  gen3684, gen3690, gen3696,
> evaluation: gen3786, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10
> evaluation: gen3788, jeremy: Re-cycle to single tank does not split, >1 tank good for maintenance.
>           pref:0.35  max:0.60  rev:0.59  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10

>>> gen4393, jeremy:  gen4394, gen4400, gen4406,
> evaluation: gen4413, roger: >1 tank good for maintenance
>           pref:0.10  max:0.10  rev:1.00  conf:1
>           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10
> evaluation: gen4415, jeremy: Re-cycle to single tank does not split, >1 tank
                                                    good for maintenance.
>           pref:0.35  max:0.60  rev:0.58  conf:1
>           pv:0.80 0.30 nil nil 1.00  uv:0.20 0.30 0.39 0.00 0.10
<> CR strategy:
                (compromise%gen275, MAIN::storageFarm%gen217)
                (MAIN::gen305 as alternative to MAIN::storageFarm%gen217)
                (consensus%gen280, MAIN::gen305)
                (majorityRule%gen281, MAIN::storageFarm%gen177)
                (compromise%gen275, MAIN::gen305)
                (MAIN::gen966 as alternative to MAIN::gen305)
                (consensus%gen280, MAIN::gen966)
                (smoothing%gen279, MAIN::storageFarm%gen177)
                (majorityRule%gen281, MAIN::storageFarm%gen217)
                (compromise%gen275, MAIN::gen966)
                (MAIN::gen2688 as alternative to MAIN::gen966)
                (consensus%gen280, MAIN::gen2688)
                (majorityRule%gen281, MAIN::gen2688)
                (smoothing%gen279, MAIN::storageFarm%gen217)
                (compromise%gen275, MAIN::gen2688)
                (MAIN::gen3683 as alternative to MAIN::gen2688)
                (consensus%gen280, MAIN::gen3683)
                (majorityRule%gen281, MAIN::gen3683)
                (smoothing%gen279, MAIN::gen305)
                (compromise%gen275, MAIN::gen3683)
                (MAIN::gen4393 as alternative to MAIN::gen3683)
                (consensus%gen280, MAIN::gen4393)
```

(majorityRule%gen281, MAIN::gen4393)
(smoothing%gen279, MAIN::gen966) <>
> accepted gen4393 <
[STORAGE gen4394
..material-of-construction mildsteel; reliability 99; unit-process-downstream MAIN::transfer%gen71;
unit-process-upstream MAIN::transfer%gen59;
(synthesisedRefinement%gen4398

>>> heatedTank%gen4456, jeremy: storage%gen4457, coil%gen4463,
> evaluation: gen4522, jeremy: Good for maintenace
> pref:0.10 max:0.10 rev:1.00 conf:1
> pv:nil nil nil nil 1.00 uv:0.20 0.30 0.40 0.00 0.10
> problem: conflict%gen4514, roger, NOT-REQUIRED HARD
> Heating coil not required
> pref:0.10 max:0.25 rev:0.40 conf:1.00
> pv:0.40 nil nil nil nil uv:0.25 0.30 0.30 0.05 0.10
> Proposal was NOT accepted
)
(selectedRefinement%gen4396

>>> storageSelProp%gen4435, jeremy: atmosTank%gen4436,
> evaluation: gen4511, roger: Not as expensive as pressure vessel
> pref:0.25 max:0.25 rev:1.00 conf:1
> pv:1.00 nil nil nil nil uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen4519, jeremy, NOT_POSSIBLE HARD
> Cannot store hot toluene in atmos. tank
> pref:0.35 max:0.40 rev:0.88 conf:0.30
> pv:nil nil 0.88 nil nil uv:0.20 0.30 0.40 0.00 0.10

>>> storageSelProp%gen4442, jeremy: floatingRoofTank%gen4443,
> evaluation: gen4512, roger: Not as expensive as pressure vessel
> pref:0.25 max:0.25 rev:1.00 conf:1
> pv:1.00 nil nil nil nil uv:0.25 0.30 0.30 0.05 0.10
> problem: conflict%gen4520, jeremy, NOT_POSSIBLE HARD
> Highly volatile and toxic fluid, floating roof not appropriate
> pref:0.00 max:0.00 rev:0.00 conf:1.00
> pv:nil nil nil nil nil uv:0.20 0.30 0.40 0.00 0.10

>>> storageSelProp%gen4449, jeremy: pressureVessel%gen4450,
> evaluation: gen4513, roger: Pressure vessel expensive
> pref:0.10 max:0.25 rev:0.40 conf:1
> pv:0.40 nil nil nil nil uv:0.25 0.30 0.30 0.05 0.10
> evaluation: gen4521, jeremy: Costly system to get tank to discharge to pressure vessel
> pref:0.00 max:0.20 rev:0.00 conf:1
> pv:0.00 nil nil nil nil uv:0.20 0.30 0.40 0.00 0.10
> accepted storageSelProp%gen4435 <
[ATMOSPHERIC_TANK atmosTank%gen4436
(synthesisedRefinement%gen4440

>>> atmosTankSynth%gen4592, jeremy: atmosTank%gen4593, blanket%gen4599,
suctionSystem%gen4605, valve%gen4611,
> evaluation: gen4618, jeremy: Valve good for removing solids; Good for maintenace
> pref:0.10 max:0.10 rev:1.00 conf:1
> pv:nil nil nil nil 1.00 uv:0.20 0.30 0.40 0.00 0.10
> accepted atmosTankSynth%gen4592 <
[ATMOSPHERIC_TANK atmosTank%gen4593
..material-of-construction mildsteel;
(parametisedRefinement%gen4594

>>> paramProp%gen4635, roger: storage%gen4637,
> accepted paramProp%gen4635 <
[ATMOSPHERIC_TANK storage%gen4637
]
)
]
[BLANKET blanket%gen4599
..unit-process-downstream MAIN::atmosTank%gen4593; unit-process-upstream

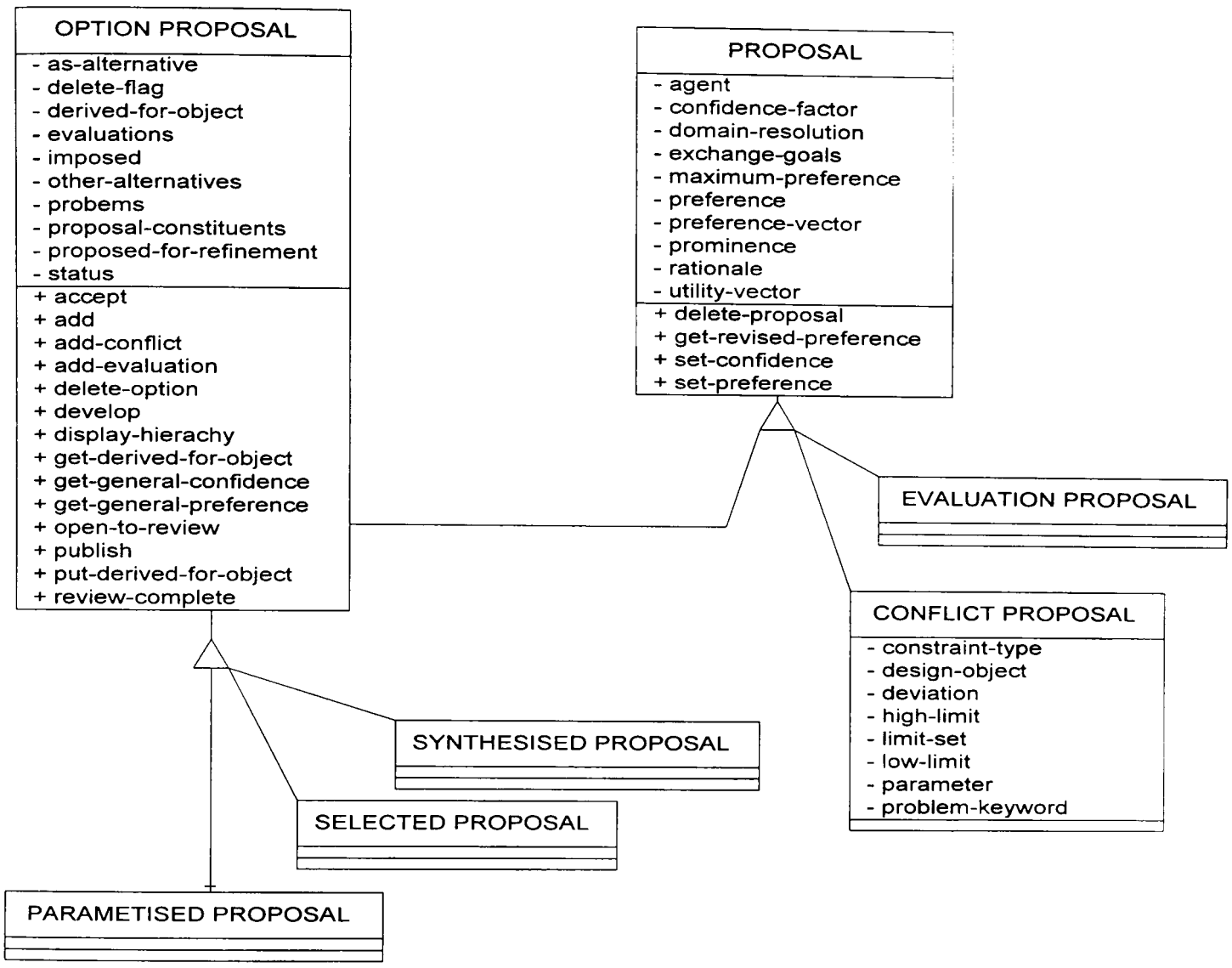
```
        MAIN::atmosTank%gen4593;
(synthesisedRefinement%gen4603

 >>> blanket%gen4646, roger:  fan%gen4647, stream%gen4653, oxidiser%gen4659,
                                                    trip%gen4665,
 > evaluation: gen4813, jeremy: Expensive;No vent with TO;Back pressure problem;
 >          pref:0.33  max:0.60  rev:0.55  conf:0.6
 >          pv:0.30 nil 0.67 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> blanket%gen4684, jeremy:  fan%gen4685, stream%gen4691, oxidiser%gen4697,
 > evaluation: gen4816, jeremy: Expensive;No vent with TO;Back pressure problem;
 >          pref:0.33  max:0.60  rev:0.55  conf:0.6
 >          pv:0.30 nil 0.67 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> blanket%gen4703, jeremy:  fan%gen4704, stream%gen4710, oxidiser%gen4716,
        vent%gen4722, trip%gen4728,
 > evaluation: gen4809, roger: Birds can next in the vents.
 >          pref:0.21  max:0.30  rev:0.70  conf:0.3
 >          pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > evaluation: gen4818, jeremy: Expensive;Backpressure with TO resolved with vent;
                  Vent open creates flam. atmos.;Back pressure problem;Vent can block;
 >          pref:0.39  max:0.60  rev:0.65  conf:0.6
 >          pv:0.30 nil 0.82 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> blanket%gen4734, jeremy:  vent%gen4735, pipe%gen4741, flare%gen4747,
 > evaluation: gen4810, roger: Birds can next in the vents.
 >          pref:0.21  max:0.30  rev:0.70  conf:0.3
 >          pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > evaluation: gen4820, jeremy: Cheaper;Flair potentially poor solution for future;
                                                    Vent can block;
 >          pref:0.52  max:0.60  rev:0.87  conf:0.5
 >          pv:0.80 nil 0.90 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> blanket%gen4753, jeremy:  vent%gen4754, pipe%gen4760, condenser%gen4766,
 > evaluation: gen4811, roger: Birds can next in the vents.
 >          pref:0.21  max:0.30  rev:0.70  conf:0.3
 >          pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > evaluation: gen4822, jeremy: Cheaper;Vent can block;
 >          pref:0.52  max:0.60  rev:0.87  conf:0.6
 >          pv:0.80 nil 0.90 nil nil  uv:0.20 0.30 0.40 0.00 0.10

 >>> blanket%gen4772, jeremy:  vent%gen4773, pipe%gen4779, atmosphere%gen4785,
 > evaluation: gen4812, roger: Birds can next in the vents.
 >          pref:0.21  max:0.30  rev:0.70  conf:0.3
 >          pv:nil nil 0.70 nil nil  uv:0.25 0.30 0.30 0.05 0.10
 > problem: conflict%gen4824, jeremy, NOT_POSSIBLE  HARD
 >        Not allowed to vent toluene to atmosphere.
 >          pref:0.00  max:0.00  rev:0.00  conf:1.00
 >          pv: uv:

 >>> gen4881, roger:  gen4882, gen4888, gen4894, gen4900,
 > evaluation: gen4911, jeremy: Expensive;No vent with TO;Back pressure problem;
 >          pref:0.30  max:0.57  rev:0.53  conf:0.6
 >          pv:0.30 nil 0.67 nil nil  uv:0.23 0.30 0.35 0.03 0.10
 <> CR strategy:
        (compromise%gen4844, MAIN::blanket%gen4646)
        (MAIN::gen4881 as alternative to MAIN::blanket%gen4646) <>
 > accepted blanket%gen4734 <
 [VENT vent%gen4735
  ..unit-process-downstream MAIN::pipe%gen4741; unit-process-upstream
        MAIN::atmosTank%gen4593;
 ]
 [PIPE pipe%gen4741
  ..unit-process-downstream MAIN::flare%gen4747; unit-process-upstream
        MAIN::vent%gen4735;
 ]
 [FLARE flare%gen4747
```

```
    ..unit-process-upstream MAIN::pipe%gen4741;
   ]
  )
]
[SUCTION_SYSTEM suctionSystem%gen4605
 ..unit-process-downstream MAIN::atmosTank%gen4593; unit-process-upstream
          MAIN::atmosTank%gen4593;
  (synthesisedRefinement%gen4609

   >>> suctionProp%gen4671, roger:  storage%gen4672, stream%gen4678,
   > problem: conflict%gen4815, jeremy, HIGH-STOCK  SOFT
   >       best not to hold stocks
   >           pref:0.39  max:0.40  rev:0.96  conf:0.80
   >           pv:nil nil 0.96 nil nil  uv:0.20 0.30 0.40 0.00 0.10

   >>> suctionSysProp%gen4791, jeremy:  suctionSystem%gen4792,
   > evaluation: gen4826, jeremy: Back breaker blow and create flammable atmosphere
   >           pref:0.25  max:0.40  rev:0.62  conf:1
   >           pv:nil nil 0.62 nil nil  uv:0.20 0.30 0.40 0.00 0.10
   > accepted suctionProp%gen4671 <
   [STORAGE storage%gen4672
    ..fluid nitrogen; reliability 100; unit-process-downstream MAIN::stream%gen4678;
    (synthesisedRefinement%gen4676

      >>> heatedTank%gen4915, jeremy:  storage%gen4916, coil%gen4923,
      > evaluation: gen4937, jeremy: Good for maintenace
      >           pref:0.10  max:0.10  rev:1.00  conf:1
      >           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
      > problem: conflict%gen4935, roger, NOT-REQUIRED  HARD
      >       Heating coil not required
      >           pref:0.10  max:0.25  rev:0.40  conf:1.00
      >           pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
      > Proposal was NOT accepted
    )
   ]
        [STREAM stream%gen4678
         ..unit-process-downstream MAIN::atmosTank%gen4593; unit-process-upstream
                MAIN::storage%gen4672;
        ]
       )
      ]
      [VALVE valve%gen4611
       ..unit-process-downstream MAIN::atmosTank%gen4593;
      ]
     )
    ]
   )
]
[STORAGE gen4400
 ..material-of-construction mildsteel; reliability 99; unit-process-upstream MAIN::pipe%gen230;
  (synthesisedRefinement%gen4404

   >>> heatedTank%gen4490, jeremy:  storage%gen4491, coil%gen4497,
   > evaluation: gen4526, jeremy: Good for maintenace
   >           pref:0.10  max:0.10  rev:1.00  conf:1
   >           pv:nil nil nil nil 1.00  uv:0.20 0.30 0.40 0.00 0.10
   > problem: conflict%gen4518, roger, NOT-REQUIRED  HARD
   >       Heating coil not required
   >           pref:0.10  max:0.25  rev:0.40  conf:1.00
   >           pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
   > Proposal was NOT accepted
  )
  (selectedRefinement%gen4402

   >>> storageSelProp%gen4469, jeremy:  atmosTank%gen4470,
   > evaluation: gen4515, roger: Not as expensive as pressure vessel
   >           pref:0.25  max:0.25  rev:1.00  conf:1
```

```
      >          pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
      > problem: conflict%gen4523, jeremy, NOT_POSSIBLE  HARD
      >       Cannot store hot toluene in atmos. tank
      >          pref:0.35  max:0.40  rev:0.88  conf:0.30
      >          pv:nil nil 0.88 nil nil  uv:0.20 0.30 0.40 0.00 0.10

      >>> storageSelProp%gen4476, jeremy:  floatingRoofTank%gen4477,
      > evaluation: gen4516, roger: Not as expensive as pressure vessel
      >          pref:0.25  max:0.25  rev:1.00  conf:1
      >          pv:1.00 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
      > problem: conflict%gen4524, jeremy, NOT_POSSIBLE  HARD
      >       Highly volatile and toxic fluid, floating roof not appropriate
      >          pref:0.00  max:0.00  rev:0.00  conf:1.00
      >          pv:nil nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10

      >>> storageSelProp%gen4483, jeremy:  pressureVessel%gen4484,
      > evaluation: gen4517, roger: Pressure vessel expensive
      >          pref:0.10  max:0.25  rev:0.40  conf:1
      >          pv:0.40 nil nil nil nil  uv:0.25 0.30 0.30 0.05 0.10
      > evaluation: gen4525, jeremy: Costly system to get tank to discharge to pressure vessel
      >          pref:0.00  max:0.20  rev:0.00  conf:1
      >          pv:0.00 nil nil nil nil  uv:0.20 0.30 0.40 0.00 0.10
      > accepted storageSelProp%gen4469 <
      [ATMOSPHERIC_TANK atmosTank%gen4470
      ]
    )
   ]
   [PIPE gen4406
    ..unit-process-downstream MAIN::storage%gen224; unit-process-upstream MAIN::storage%gen218;
   ]
  )
]
[TRANSFER transfer%gen71
 ..unit-process-downstream MAIN::still; unit-process-upstream MAIN::storage%gen65;
]
  [TRANSFER recycle%gen77
   ..fluid-contaminants particulates; unit-process-downstream MAIN::storage%gen65;
                      unit-process-upstream MAIN::still;
    (parametisedRefinement%gen78

      >>> transParamProp%gen113, jeremy:  transfer%gen114,
      > accepted transParamProp%gen113 <
      [TRANSFER transfer%gen114
      ]
    )
   ]
  )
]
```

Appendix K. CDEX Detailed design

Proposals



PROPOSAL

A proposal contains design elements that a particular agent considers are best to progress the current design. It is assumed that the agents are rational.

When a proposal is made for an object, all agents that can review the object, and its parent classes should be called to review the proposal. e.g. pump proposed, therefore review the object as a pump, and an equipment type. This is necessary so that is an object changes any parent parameters, the parent parameters must be reviewed.

Name: agent [PROPOSAL.]
Type: Attribute
Description:
Name of the agent who presented the proposal.

Name: confidence-factor [PROPOSAL.]

Type: Attribute
Description:
A confidence factor is related to the probability but it is used as a fuzzy term. It is really the confidence that an agent places on the design actually turning out the way it has proposed (or as successful - defined by the preference).

It depends on factors such as: is it a standard design which will be used wherever possible; is it novel - may be optimal, but possibly likely to be modified because of cost for example; it is a simple problem and the match is quite straight forward; it is a complex problem, and this may be the way to do it but uncertain etc.

Fuzzy variables are therefore: complete confidence; very confident; most likely; possibly; uncertain; unlikely. There should not be anything at a lower scale than unlikely, as they should not propose it as a proposal in the first place.

Name: domain-resolution [PROPOSAL.]
Type: Attribute
Description:
A list of domain specific conflict resolution strategies that can be applied by the agent presenting a view. An empty list indicates that no strategies are available from this agent for this proposal.

Name: exchange-goals [PROPOSAL.]
Type: Attribute
Description:
This slot indicates that an agent can review its design proposal given criteria specified in the design or evaluate argument. This is not an agent property as in some cases the agent may not have the necessary capability.

Name: maximum-preference [PROPOSAL.]
Type: Attribute
Description:
An indication of the maximum preference attainable from the utility function given the case that only a few goals are considered. Specified by the agent.

Default is 1 (all goals considered).

Name: preference [PROPOSAL.]
Type: Attribute
Description:
Preference is a fuzzy term that is used to indicate how desirable the proposal is to the specified agent.

The term is derived from the agents goal hierarchy.

Name: preference-vector [PROPOSAL.]
Type: Attribute
Description:
The vector that represents to an agent, how much of each goal (or issue) was resolved. This is different from the utility vector which highlights which issue is important. A combination of these vectors will determine the preference of a proposal.

Name: prominence [PROPOSAL.]
Type: Attribute
Description:
The prominence is used to denote the difference between the preference of this proposal put forward by an agent, and its next best. A higher prominence signifies that other solutions requested to be put forward by the agent will be less acceptable (and therefore the agent will be less flexible in concession making).

The default value is nil, which signifies that no prominence has been specified. Prominence is normally specified by an agent that can perform a selection process, and may know its 'next best' proposal.

Name: rationale [PROPOSAL.]
Type: Attribute
Description:
The 'rationale' is where text can be specified by an engineer to enable the rationale for particular design decision and proposal to be recorded. Rationale is really the users thoughts and is therefore usually difficult to include - the system can however be extended to enable the engineer to specify the links to the information that was used in his evaluation and generation of the design.

Name: utility-vector [PROPOSAL.]
Type: Attribute
Description:
The utility vector, or weightings of the goals that the agent considers important and used to determine how good the proposal is (preference).

Name: delete-proposal [PROPOSAL.]
Type: Operation
Description:
Delete the proposal. Nothing else is deleted. Proposals that contain design objects are deleted as part of the option proposal. All objects that have instance lists test whether the objects still exist (instance-existp) before reviewing them.

Name: get-revised-preference [PROPOSAL.]
Type: Operation
Description:

Calculate and return the revised preference for the proposal. The revised preference is the percentage of the preference to the maximum preference expressed as a number between 0 and 1. It is important as we wish to determine whether a low preference is due to only non-critical values being considered for example.

Name: set-confidence [PROPOSAL.]
Type: Operation
Description:
Set 'confidence' of a proposal

Name: set-preference [PROPOSAL.]
Type: Operation
Description:
Set 'preference' of a proposal.

OPTION PROPOSAL

An option proposal is used where an agent requires to progress the design either by proposing a refinement (selection), a decomposition (synthesis), or sizing an object (parametric design).

Name: as-alternative [OPTION PROPOSAL.]
Type: Attribute
Description:
Boolean member variable that denotes whether the proposal was put forward as an alternative. This is important for the refinement object, as an EVALUATION will not be requested if the proposal is an alternative, as this will be left to the negotiation control mechanism.

Name: delete-flag [OPTION PROPOSAL.]
Type: Attribute
Description:
Indicates whether the option proposal has been requested to be deleted. Default is FALSE.

Name: derived-for-object [OPTION PROPOSAL.]
Type: Attribute
Description:
A record of the immediate parent object, i.e. which one the agent is designing for. Value recorded when 'Proposal n for X', X being the design object.

Name: evaluations [OPTION PROPOSAL.]
Type: Attribute

Description:
When a design object is EVALUATED the agents will respond with evaluation proposals. These evaluation proposals are recorded here for the given options proposal. A check is made to ensure that the argument passed if of type EVALUATION PROPOSAL.

Name: imposed [OPTION PROPOSAL.]
Type: Attribute
Description:
Imposed indicates whether the proposal should be chosen over all other proposals. This is so that an engineer can impose a solution for test purposes so that the design can be analysed and progressed. The default is always false.

Name: other-alternatives [OPTION PROPOSAL.]
Type: Attribute
Description:
Denotes whether other alternatives are available. An agent will set this value to TRUE when it has a list of alternatives (probably ordered by preference) and it could provide another alternative if requested.

Name: problems [OPTION PROPOSAL.]
Type: Attribute
Description:
The problems are a list of conflicts that have been identified by agents for the proposal in question. Conflicts can be identified at any time.

Name: proposal-constituents [OPTION PROPOSAL.]
Type: Attribute
Description:
A proposal consists of design objects. The number of objects is dependent on the type of design process applied as follows:

if synthesis process, proposal has many design objects if selection process, proposal has a single object if parametric process, proposal has a single object

Name: proposed-for-refinement [OPTION PROPOSAL.]
Type: Attribute
Description:
A pointer to the parent REFINEMENT object. Records the refinement for which the proposal is involved (the refinement will hold other competing proposals).

Name: status [OPTION PROPOSAL.]
Type: Attribute
Description:
Indicates the status of a proposal. One of: DEFINITION, PROPOSED,

AGENT_REVIEW_COMPLETE, or ACCEPTED

Name: accept [OPTION PROPOSAL.]
Type: Operation
Description:
Message sent when the proposal is accepted. The proposal is accepted if it is the most suitable proposal made (considering the global perspective) or it is the only proposal and there are no hard conflicts.

The proposal is not deleted when accepted as we need to keep links to slots elsewhere in the hierarchy that were involved in the design of the proposal in the first place.

Accept will call accept() for each of the design objects in the proposal. It is important to inform the design objects that they have been accepted so that they can request further design to be performed.

When a parametised proposal is accepted, its values are transferred to the parents.

Name: add [OPTION PROPOSAL.]
Type: Operation
Description:
Add an item to a proposal (proposal constituents). The 'design-by-process' slot in the design object is set depending on the type of proposal created (parametised, selected, or synthesised). The proposal reference is assigned to the design object slot parent-proposal.

Name: add-conflict [OPTION PROPOSAL.]
Type: Operation
Description:
Adds a conflict to proposal 'problems'. A check is made to ensure the a CONFLICT PROPOSAL is passed as an argument.

Name: add-evaluation [OPTION PROPOSAL.]
Type: Operation
Description:
Add an evaluation to the 'evaluations'. This method is called whenever an evaluation of the proposal is presented in the design.

Name: delete-option [OPTION PROPOSAL.]
Type: Operation
Description:
Delete a proposal. This requires:

- deleting all associated evaluation proposals (evaluations) - deleting all problems conflicts identified (problems) - deleting all objects that make up the proposal

(proposal-constituents)

Finally the proposal will be deleted itself.

Name: develop [OPTION PROPOSAL.]
Type: Operation
Description:
Develop is the method of enabling a proposal to be developed (further refined) without acceptance. It is called from the negotiation mechanism in order to attain more reliability in its assessments of a design approach where there is uncertainty.

Name: display-hierarchy [OPTION PROPOSAL.]
Type: Operation
Description:
Display information regarding the proposal (evaluations, conflicts etc). This information is sent to the logical router maintained by the design object. This method can only be called indirectly through the design object.

Name: get-derived-for-object [OPTION PROPOSAL.]
Type: Operation
Description:
Returns derived-for-object. (see derived-for-object)

Name: get-general-confidence [OPTION PROPOSAL.]
Type: Operation
Description:
Determine a general confidence for the proposal from the perspective of the proposal and its respective evaluations and conflicts.

C confidence of a proposal is not included in the calculation if the max preference is 0 (i.e. a confidence in no opinion).

The calculation assumes the following: If agent A is highly confident of proposal, and agent B is not highly confident, then one does not want the confidence to be as high as A's, and inverselyone does not want the confidence to be as low as B's. A function to take the average would resolve the above problem.

It may be considered that there is another factor at play. Given that agent A may only have viewed the problem from a very small perspective from his total values (i.e. low max preference), and agent B (with the low confidence) viewed the solution using all his values (high max preference), should agent B's confidence have more weight (is he more informed)? This however is not considered important as Agent A may not think any less of the total proposal, it is just that only a small part of the proposal concerns him.

Name: get-general-preference [OPTION PROPOSAL.]
Type: Operation
Description:
Determine a general preference for the proposal by reviewing the preferences for the proposal, evaluations and conflicts.

An average is taken of revised preferences, the revised preferences being the percentage of proposal preference to maximum-preference of a proposal.

The average is used as it models the following qualitative assumptions: If agent A prefers a proposal to agent B, the total preference should not be as high as agent A's but should not be as low as agent B's.

Name: open-to-review [OPTION PROPOSAL.]
Type: Operation
Description:
'open-to-review' is a message indicating that the option proposal has been put out for review by other agents. It is essentially an indicator for the proposal to ensure that no other objects are added to the proposal after it has been made open for review purposes.

Name: publish [OPTION PROPOSAL.]
Type: Operation
Description:
Display information regarding the proposal. All objects that make up the proposal are requested to report on their structure/information content.

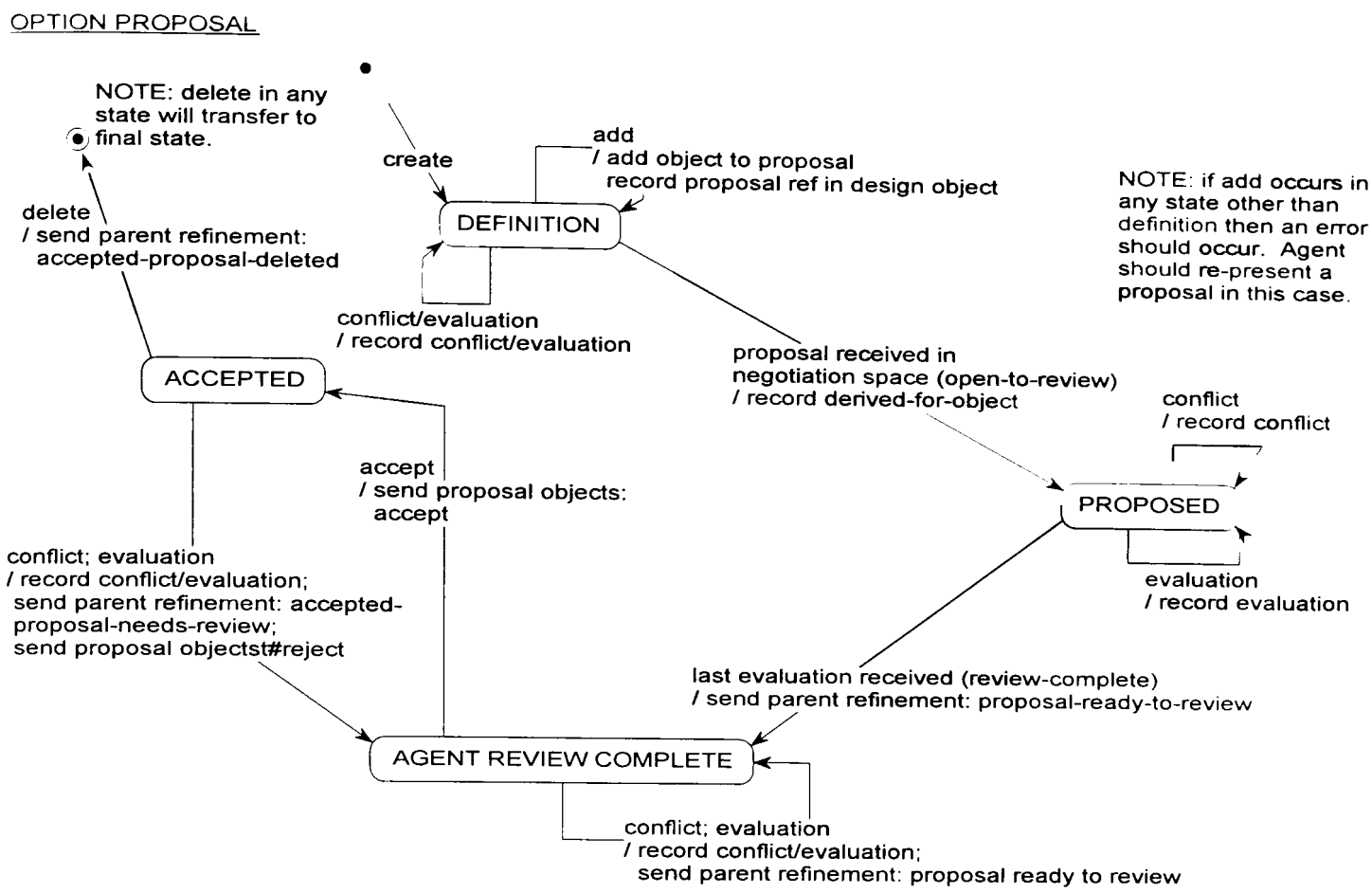
Name: put-derived-for-object [OPTION PROPOSAL.]
Type: Operation
Description:
Records the immediate parent object, i.e. which one the agent is designing for. Value recorded when 'Proposal n for X', X being the design object.

For each object in proposal-constituents, the derived-from-object is set to the specified design object. Additionally, If the derived-for-object was designed-by-process PARAMETRIC, and this is a parametric proposal, then the parent-design-object for each object in the proposal-constituents is set to the parent-design-object of the derived-for-object. If not, the parent-design-object for each object in the proposal-constituents is set to the derived-for-object.

Name: review-complete [OPTION PROPOSAL.]
Type: Operation
Description:
A message sent to indicate that all reviews of the proposal have been made by those agents that can. The proposal is then ready for review.

The parent refinement is informed: proposal-ready-to-review.

State transition for OPTION PROPOSAL



State: DEFINITION

Description:

Represents the state where an agent is adding design objects to a proposal. The agent can present conflicts and evaluations in this phase of what he considers is appropriate.

State: PROPOSED

Description:

Proposed is a state of a proposal where it is waiting for all the evaluations that were REQUESTED to complete.

This is a safeguard to prevent design from continuing without at least one review by each of the interested agents. If agents consider it necessary in the future to present conflicts and evaluations on the proposal then they can do, this however may lead to design re-work.

State: AGENT REVIEW COMPLETE

Description:

A state where a proposal has been reviewed by those agents requested to evaluate a proposal. This does not indicate that agents can not longer present conflicts or evaluations. just that the proposal is ready to be accounted for in the design if necessary.

If agents present conflicts or evaluations at this stage, the problem is not serious. There may have been a better route for the design to take but the proposal was not accepted as a basis to proceed design.

State: ACCEPTED
Description:
The proposal has been accepted by the agent as the best approach for the design to take from considering the global perspective.

If evaluations and conflict occur in this state then the problem may be serious as the accepted design so far may be invalid.

EVALUATION PROPOSAL

An evaluation proposal is provided by an agent as a result of a request to evaluate a proposal. An evaluation is treated as a proposal as it is (or can be) dependent upon other design information (and therefore consistency needs to be maintained) and is a view expressed by an agent.

CONFLICT PROPOSAL

A conflict is treated as a proposal as it is a viewpoint by an agent (albeit regarding another proposal) that is/can be dependent upon other information in the design. Conflicts can be removed from consideration given that the design information on which the conflict was accessed is changed.

Name: constraint-type [CONFLICT PROPOSAL.]
Type: Attribute
Description:
A conflict can be due to violation of a HARD or a SOFT constraint. Hard constraints usually lead to immediate rejection of a proposal - depending on confidence (likelihood of constraint being a problem). Soft constraints can be compromised.

Allowed symbols: HARD, SOFT. Default SOFT.

Name: design-object [CONFLICT PROPOSAL.]
Type: Attribute
Description:
A conflict may be due to the/an object in the proposal and therefore the design object should be specified. If the conflict is not due to the objects but the proposal itself then the design-object is left null.

Name: deviation [CONFLICT PROPOSAL.]
Type: Attribute
Description:

Refers to the deviation of the parameter specified in the conflict proposal. This may be INRANGE, OUTRANGE, HIGH or LOW if the offending slot is numerical, otherwise MEMBER or NOT-MEMBER if the slot contains one of a set of values.

The range is determined by high-limit and low-limit variables. The member functionality is determined by limit-set. If HIGH or LOW deviation used, high refers to the number higher than high-limit, and LOW refers to the number in low-limit.

Name: high-limit [CONFLICT PROPOSAL.]
Type: Attribute
Description:
If an offending slot is numerical, then this slot may contain a value that denotes the high end of a range if the deviation is either INRANGE or OUTRANGE, or contains the upper limit if the deviation is HIGH.

Name: limit-set [CONFLICT PROPOSAL.]
Type: Attribute
Description:
Holds the set of limiting values for a design parameter. Used in conjunction with the 'member' deviations (notmember, member).

Name: low-limit [CONFLICT PROPOSAL.]
Type: Attribute
Description:
The low limit denotes the low end of a range if the conflict is over a numerical slot and the deviation is either INRANGE or OUTRANGE. If the deviation is LOW, then this low-limit denotes the lower limit that the slot should not fall below.

Name: parameter [CONFLICT PROPOSAL.]
Type: Attribute
Description:
Refers to the name of the offending slot if the conflict refers to a specific slot and problem keyword not specified.

Name: problem-keyword [CONFLICT PROPOSAL.]
Type: Attribute
Description:
A problem-keyword denotes the type of problem. These keywords can be standardised for the types of problem that a device may have (possibly derived in HAZOP style). These keywords can be used to provide a standard language for facilitating the development of standard conflict resolution mechanisms. The agent presents his capabilities to resolve domain conflicts by specifying the problem-keyword(s) in the domain-resolution list when presenting the proposal.

PARAMETISED PROPOSAL

An OPTION_PROPOSAL containing the parametric design of a design object. The proposal can only contain one design object. See OPTION_PROPOSAL.

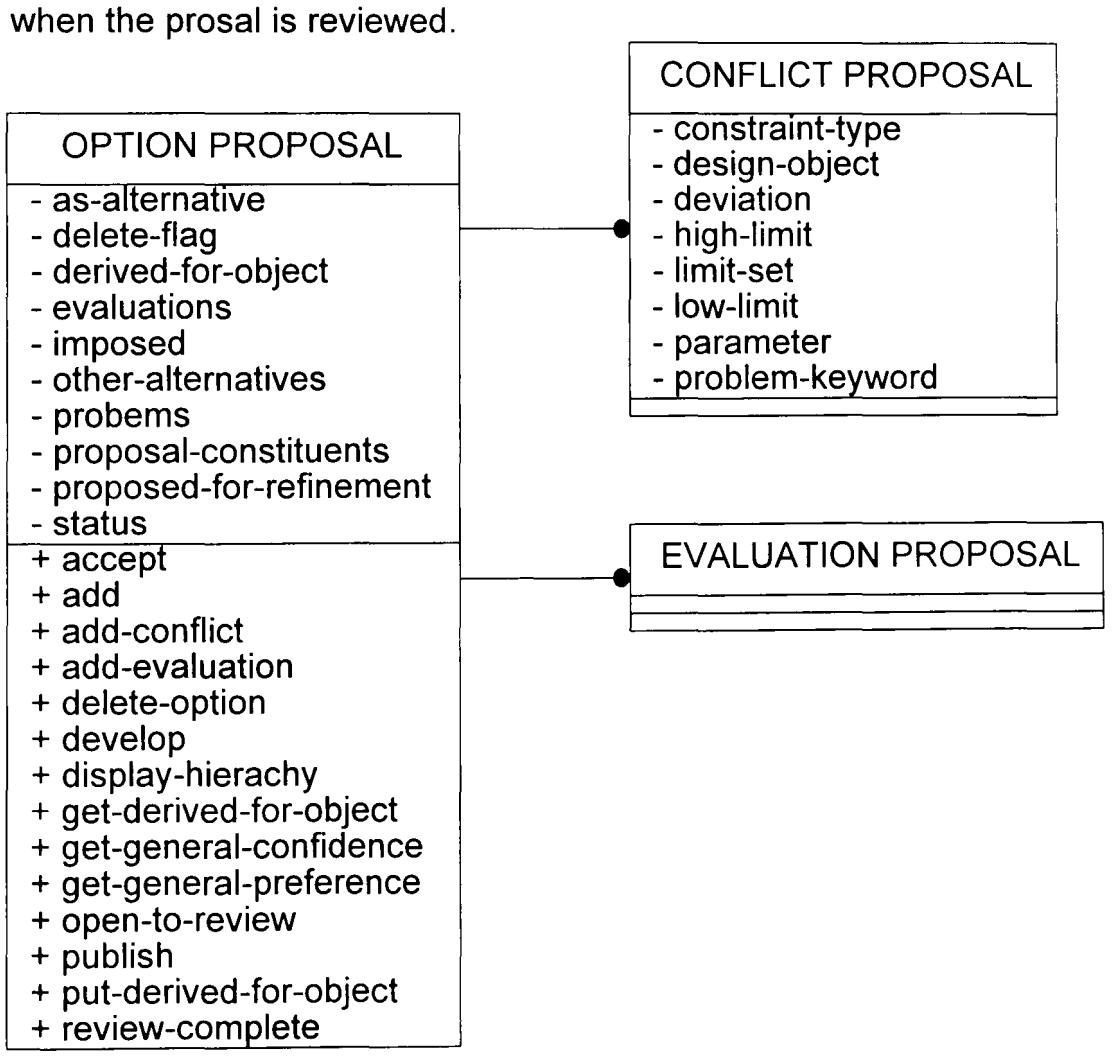
SELECTED PROPOSAL

An OPTION_PROPOSAL containing the selection of a design object. The proposal can contain only one design object. See OPTION_PROPOSAL.

SYNTHESISED PROPOSAL

An OPTION_PROPOSAL containing the synthesis of a design object. The proposal can contain many design objects. See OPTION_PROPOSAL.

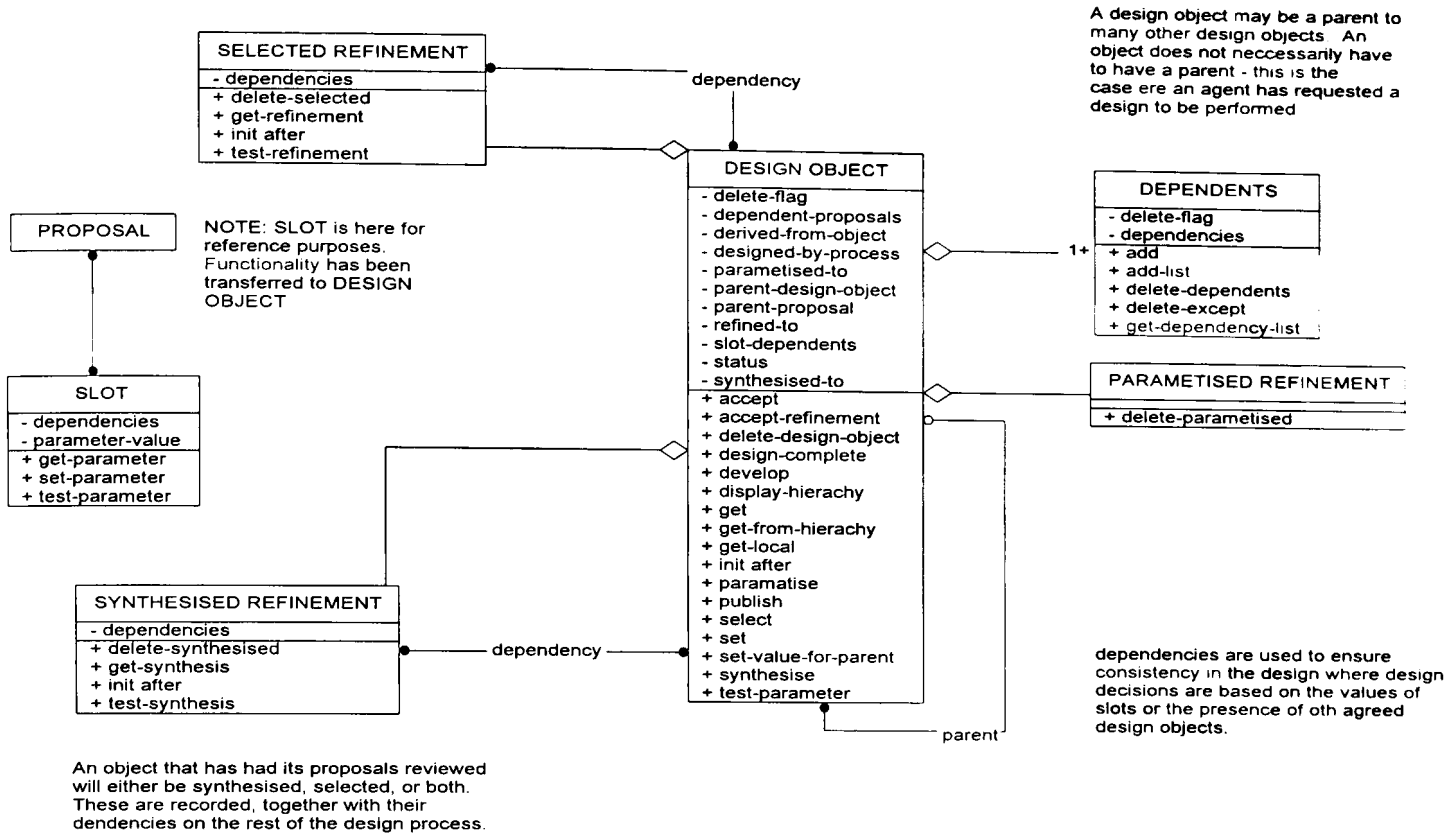
Only proposals that identify design objects can have conflicts associated with them. The conflicts are reviewed at the time when the prosal is reviewed.



This object diagram depicts the relationships between the option proposal and the conflict and evaluation proposals. Although they are all of type proposal - due to their common characteristic of being dependent upon other design information - an option proposal may

have many conflicts and evaluation proposals. These conflict and evaluation proposals denote an agents level of support for a particular option (option proposal) put forward by another agent.

Design Object



DESIGN OBJECT

A design object is the parent class of all items that can be part of the design in the negotiated environment. It is the basic object in the environment and is responsible for setting up the different types of design refinements (selection, synthesis, parametric) that manage proposals and request conflict resolution from the negotiation mechanism. It manages access to the objects slots (attributes) and records dependencies between the object and other parts of the design.

When a design object is created, all the values are by default set to 'null'. This enables us to track which variables have been set by an agent when a proposal is made. A 'null' value for a slot does not mean that the slot does not have a value, the slot may be a member of one of its parent objects if the object was derived as a result of a selection or parametric design process and have a value and associated dependencies.

Name: delete-flag [DESIGN OBJECT.]
Type: Attribute
Description:
Indicates whether the object is part of a delete chain. Default is FALSE. The object cannot be deleted without the objects children (other design refinements, objects etc) being deleted first.

Name: dependent-proposals [DESIGN OBJECT.]
Type: Attribute
Description:

A proposal may be dependent on a design object. For example, when a parametric design is accepted and its local values are passed up to parent design object. then the parent design object must then be dependent on the object that was proposed.

Name: derived-from-object [DESIGN OBJECT.]
Type: Attribute
Description:
'derived-from-object' is the first parent of the current design object. It is different from parent-design-object in that the parent-design-object for a parametric design records the first parent in the chain (the object at the start of the synthesis process). The derived-from-object in the synthesis and selection process is exactly the same the parent-design-object.

The value is important in the get-from-hierarchy function where the immediate parent in a parametric design chain is important.

Name: designed-by-process [DESIGN OBJECT.]
Type: Attribute
Description:
Indicates how the object was designed, by a synthesis, selection, or parametric design process.

Values: SYNTHESIS, SELECTION, PARAMETRIC

Value used in the Get() function, to determine whether a parent would exist with a shared slot. Slots can only be shared in a SELECTION, or PARAMETRIC design process. An object that is a result of a synthesis process has no parents with similar slots.

Name: parametised-to [DESIGN OBJECT.]
Type: Attribute
Description:
A pointer to the refinement object that contains proposals made for the parametric design of the current object.

Name: parent-design-object [DESIGN OBJECT.]
Type: Attribute
Description:
A design object keeps track of its parent design object.

- If object synthesised, the parent is the immediate object from which it was synthesised.

- if the object was parametised, the parent is the object that is at the start of the parametric design process (no record of intermediate stages).

- if object selected, parent is the immediate parent from which it was selected.

Name: parent-proposal [DESIGN OBJECT.]
Type: Attribute
Description:
A pointer to the proposal in which the object was proposed. An object can only belong to a single proposal.

Default is nil, as in the case of the ORIGIN object, where there is no parent proposal.

Name: refined-to [DESIGN OBJECT.]
Type: Attribute
Description:
A pointer to the refinement object that contains a list of proposals put forward as a 'selection' of the current design object.

Name: slot-dependents [DESIGN OBJECT.]
Type: Attribute
Description:
A list of slots together with their dependencies. slot-dependents is a multislot organised in a 'slot [DEPENDENTS] slot [DEPENDENTS] ...' fashion.

Name: status [DESIGN OBJECT.]
Type: Attribute
Description:
Design status of the object:

can be one of: CREATED, SIZING, REFINEMENT, REFINED

Name: synthesised-to [DESIGN OBJECT.]
Type: Attribute
Description:
A pointer to a refinement object that contains proposals put forward as a design synthesis of the current object.

Name: accept [DESIGN OBJECT.]
Type: Operation
Description:
When a proposal is accepted, each object in that proposal becomes available for design. If the object was a result of a synthesis process then there are no parent attributes that are shared with the current local attributes and therefore object status just becomes accepted.

If the object was derived from a selection or parametric design process, then there may be attributes that are shared between the parent and child objects. These attributes therefore

have to be propagated up to the parents and associated dependency changes accounted for. If a parametric or selected design is accepted, then obviously one is accepting all the parents up to the starting point of this particular design process. e.g. if pump103 is selected which is a centrifugal pump, which is a pump, then one is obviously also accepting its characteristics. In a parametric design, if one accepts a design that is derived from a whole sequence of calculations, then one is accepting all the proposals that they are based on etc. This is not the case with design synthesis, where one may accept a particular pump configuration, but one is not accepting the complete design synthesis.

An object can be accepted that is already accepted, this is because it is possible for some parameters to change in a proposal for say further selection, that have to be reflected in the parent objects.

Name: accept-refinement [DESIGN OBJECT.]
Type: Operation
Description:
The message accept-refinement is sent to the design object when one of its refinements (parametric, selection, or synthesis) is accepted. See State Transition Diagram (STD). A check is performed to see if the design object has been completely designed and therefore can be further designed.

Name: delete-design-object [DESIGN OBJECT.]
Type: Operation
Description:
Delete the design-object.

All slot dependencies, design refinements and synthesis objects, and dependencies on the refinements and synthesis should also be deleted and/or back open for review.

Slots deleted: dependent-proposals; parametised-to; slot dependents; refined-to; synthesised-to;

Name: design-complete [DESIGN OBJECT.]
Type: Operation
Description:
A message indicating that one of the refinements (parametric, selection, synthesis) has been completed. See STD.

Name: develop [DESIGN OBJECT.]
Type: Operation
Description:
A request to 'develop' an object, that is to size and refine. Issued by the negotiation mechanism. See STD. When design is accepted, an accept message is sent which is similar to develop in the way in which the state is modified, but the difference is where the objects

slots become 'accepted' and are therefore transferred to the parent objects if necessary.

Name: display-hierarchy [DESIGN OBJECT.]
Type: Operation
Description:
Display a hierarchy, starting from the current object, and progressing down to the refinement and proposals. Only the 'accepted' proposals are shown in the hierarchy.

Name: get [DESIGN OBJECT.]
Type: Operation
Description:
A general form of the get functions that returns a value of a slot and records the proposal dependency with that slot. There are two forms of get, depending on how the object was designed (designed-by-process).

Name: get-from-hierarchy [DESIGN OBJECT.]
Type: Operation
Description:
Return value of a slot. This Get() function is called if the object was a result of a selection or parametric design process (i.e. has shared attributes with parent). The proposal dependent upon the value of the slot is specified so that a dependency link can be maintained.

The value of a slot may be null, but if the slot is also local to the derived-from-object, then that may hold the value. Note that a derived-form-object will contain the value only if the object was designed by a selection, or parametric design process.

If proposal is not specified ('null') then the dependency is not recorded.

Name: get-local [DESIGN OBJECT.]
Type: Operation
Description:
Get a local value of a slot. This Get function is called when the design object does not share any attributes with its parents and therefore there is no need for interrogation of parent attributes if value is not available.

This function is called if the object is designed-by-process synthesis. The alternate form of this function is get-from-hierarchy.

Name: init after [DESIGN OBJECT.]
Type: Operation
Description:
On initialisation, the object must create the objects PARAMETISED REFINEMENT, SYNTHESISED REFINEMENT, and SELECTED REFINEMENT. The object pointer

is passed to the reference object.

Name: paramatise [DESIGN OBJECT.]
Type: Operation
Description:
The design object has been parametised and a proposal is presented. Note that more than one proposal can exist for a parametric design (although probably unusual).

Name: publish [DESIGN OBJECT.]
Type: Operation
Description:
Display information about the design object and also request further design detail in the proposals to be published.

Name: select [DESIGN OBJECT.]
Type: Operation
Description:
Inform the design object that another design object has been selected based on its requirements (a proposal).

Name: set [DESIGN OBJECT.]
Type: Operation
Description:
Set a value of a slot. The value is set locally to the object, even though it may be a parameter that is also local to its parent if it was involved in a selection or parametric design process.

If the slot has dependent proposals then these will be removed.

Name: set-value-for-parent [DESIGN OBJECT.]
Type: Operation
Description:
Objects may share slots with its parents if the object has been selected or involved in a parametric design process. Therefore if a value is set, then it has to be assigned to the object where the slot is ensured to be consistent throughout its design (i.e. where the slot is first defined in a type hierarchy). If a value is null in an object it does not mean there is no value assignment, if the slot also belongs to a parent then the parent may store the value. 'set-value-for-parent' works in conjunction with get.

If the slot has dependencies and the value has been set, then the dependencies to that slot must be deleted as well. If the design has been accepted, then the 'cost-of-change', i.e. the effect of changing the slot, should have been accounted for.

A check is made if the parent-design-object is null in the case where the object is the first

in the hierarchy, i.e. the origin object.

Name: synthesise [DESIGN OBJECT.]
Type: Operation
Description:
Inform the design object that it has been synthesised to a set of specified design objects (a proposal).

Name: test-parameter [DESIGN OBJECT.]
Type: Operation
Description:
Returns true if the slot has been assigned a value. This is valuable in a design process, as an agent needs to be able to test if the appropriate values are available to perform the design BEFORE recording any dependency with the slot. If the object was derived from a parametric design process, then the slot value may be high in the hierarchy, and therefore we should test for parent values not being null etc.

Type: Class
Description:
A design object is the parent class of all items that can be part of the design in the negotiated environment. It is the basic object in the environment and is responsible for setting up the different types of design refinements (selection, synthesis, parametric) that manage proposals and request conflict resolution from the negotiation mechanism. It manages access to the objects slots (attributes) and records dependencies between the object and other parts of the design.

When a design object is created, all the values are by default set to 'null'. This enables us to track which variables have been set by an agent when a proposal is made. A 'null' value for a slot does not mean that the slot does not have a value, the slot may be a member of one of its parent objects if the object was derived as a result of a selection or parametric design process and have a value and associated dependencies.

Name: delete-flag [DESIGN OBJECT.]
Type: Attribute
Description:
Indicates whether the object is part of a delete chain. Default is FALSE. The object cannot be deleted without the objects children (other design refinements, objects etc) being deleted first.

Name: dependent-proposals [DESIGN OBJECT.]
Type: Attribute
Description:
A proposal may be dependent on a design object. For example, when a parametric design is accepted and its local values are passed up to parent design object, then the parent design

object must then be dependent on the object that was proposed.

Name: derived-from-object [DESIGN OBJECT.]
Type: Attribute
Description:
'derived-from-object' is the first parent of the current design object. It is different from parent-design-object in that the parent-design-object for a parametric design records the first parent in the chain (the object at the start of the synthesis process). The derived-from-object in the synthesis and selection process is exactly the same the parent-design-object.

The value is important in the get-from-hierarchy function where the immediate parent in a parametric design chain is important.

Name: designed-by-process [DESIGN OBJECT.]
Type: Attribute
Description:
Indicates how the object was designed, by a synthesis, selection, or parametric design process.

Values: SYNTHESIS, SELECTION, PARAMETRIC

Value used in the Get() function, to determine whether a parent would exist with a shared slot. Slots can only be shared in a SELECTION, or PARAMETRIC design process. An object that is a result of a synthesis process has no parents with similar slots.

Name: parametised-to [DESIGN OBJECT.]
Type: Attribute
Description:
A pointer to the refinement object that contains proposals made for the parametric design of the current object.

Name: parent-design-object [DESIGN OBJECT.]
Type: Attribute
Description:
A design object keeps track of its parent design object.

- If object synthesised, the parent is the immediate object from which it was synthesised.

- if the object was parametised, the parent is the object that is at the start of the parametric design process (no record of intermediate stages).

- if object selected, parent is the immediate parent from which it was selected.

Name: parent-proposal [DESIGN OBJECT.]
Type: Attribute
Description:
A pointer to the proposal in which the object was proposed. An object can only belong to a single proposal.

Default is nil, as in the case of the ORIGIN object, where there is no parent proposal.

Name: refined-to [DESIGN OBJECT.]
Type: Attribute
Description:
A pointer to the refinement object that contains a list of proposals put forward as a 'selection' of the current design object.

Name: slot-dependents [DESIGN OBJECT.]
Type: Attribute
Description:
A list of slots together with their dependencies. slot-dependents is a multislot organised in a 'slot [DEPENDENTS] slot [DEPENDENTS] ...' fashion.

Name: status [DESIGN OBJECT.]
Type: Attribute
Description:
Design status of the object:

can be one of: CREATED, SIZING, REFINEMENT, REFINED

Name: synthesised-to [DESIGN OBJECT.]
Type: Attribute
Description:
A pointer to a refinement object that contains proposals put forward as a design synthesis of the current object.

Name: accept [DESIGN OBJECT.]
Type: Operation
Description:
When a proposal is accepted, each object in that proposal becomes available for design. If the object was a result of a synthesis process then there are no parent attributes that are shared with the current local attributes and therefore object status just becomes accepted.

If the object was derived from a selection or parametric design process, then there may be attributes that are shared between the parent and child objects. These attributes therefore have to be propagated up to the parents and associated dependency changes accounted for. If a parametric or selected design is accepted, then obviously we are accepting all the

parents up to the starting point of this particular design process. e.g. if pump103 is selected which is a centrifugal pump, which is a pump, one is also accepting its characteristics. In a parametric design, if one accepts a design that is derived from a whole sequence of calculations, then one is accepting all the proposals that they are based on etc. This is not the case with design synthesis, where one may accept a particular pump configuration, but without accepting the complete design synthesis.

An object can be accepted that is already accepted, this is because it is possible for some parameters to change in a proposal for say further selection, that have to be reflected in the parent objects.

Name: accept-refinement [DESIGN OBJECT.]
Type: Operation
Description:
The message accept-refinement is sent to the design object when one of its refinements (parametric, selection, or synthesis) is accepted. See State Transition Diagram (STD). A check is performed to see if the design object has been completely designed and therefore can be further designed.

Name: delete-design-object [DESIGN OBJECT.]
Type: Operation
Description:
Delete the design-object.

All slot dependencies, design refinements and synthesis objects, and dependencies on the refinements and synthesis should also be deleted and/or back open for review.

Slots deleted: dependent-proposals; parametised-to; slot dependents; refined-to; synthesised-to;

Name: design-complete [DESIGN OBJECT.]
Type: Operation
Description:
A message indicating that one of the refinements (parametric, selection, synthesis) has been completed. See STD.

Name: develop [DESIGN OBJECT.]
Type: Operation
Description:
A request to 'develop' an object, that is to size and refine. Issued by the negotiation mechanism. See STD. When design is accepted, an accept message is sent which is similar to develop in the way in which the state is modified, but the difference is where the objects slots become 'accepted' and are therefore transferred to the parent objects if necessary.

Name: display-hierarchy [DESIGN OBJECT.]
Type: Operation
Description:
Display a hierarchy, starting from the current object, and progressing down to the refinement and proposals. Only the 'accepted' proposals are shown in the hierarchy.

Name: get [DESIGN OBJECT.]
Type: Operation
Description:
A general form of the get functions that returns a value of a slot and records the proposal dependency with that slot. There are two forms of get, depending on how the object was designed (designed-by-process).

Name: get-from-hierarchy [DESIGN OBJECT.]
Type: Operation
Description:
Return value of a slot. This Get() function is called if the object was a result of a selection or parametric design process (i.e. has shared attributes with parent). The proposal dependent upon the value of the slot is specified so that a dependency link can be maintained.

The value of a slot may be null, but if the slot is also local to the derived-from-object, then that may hold the value. Note that a derived-form-object will contain the value only if the object was designed by a selection, or parametric design process.

If proposal is not specified ('null') then the dependency is not recorded.

Name: get-local [DESIGN OBJECT.]
Type: Operation
Description:
Get a local value of a slot. This Get function is called when the design object does not share any attributes with its parents and therefore there is no need for interrogation of parent attributes if value is not available.

This function is called if the object is designed-by-process synthesis. The alternate form of this function is get-from-hierarchy.

Name: init after [DESIGN OBJECT.]
Type: Operation
Description:
On initialisation, the object must create the objects PARAMETISED REFINEMENT, SYNTHESISED REFINEMENT, and SELECTED REFINEMENT. The object pointer is passed to the reference object.

Name: paramatise [DESIGN OBJECT.]
Type: Operation
Description:
The design object has been parametised and a proposal is presented. Note that more than one proposal can exist for a parametric design (although probably unusual).

Name: publish [DESIGN OBJECT.]
Type: Operation
Description:
Display information about the design object and also request further design detail in the proposals to be published.

Name: select [DESIGN OBJECT.]
Type: Operation
Description:
Inform the design object that another design object has been selected based on its requirements (a proposal).

Name: set [DESIGN OBJECT.]
Type: Operation
Description:
Set a value of a slot. The value is set locally to the object, even though it may be a parameter that is also local to its parent if it was involved in a selection or parametric design process.

If the slot has dependent proposals then these will be removed.

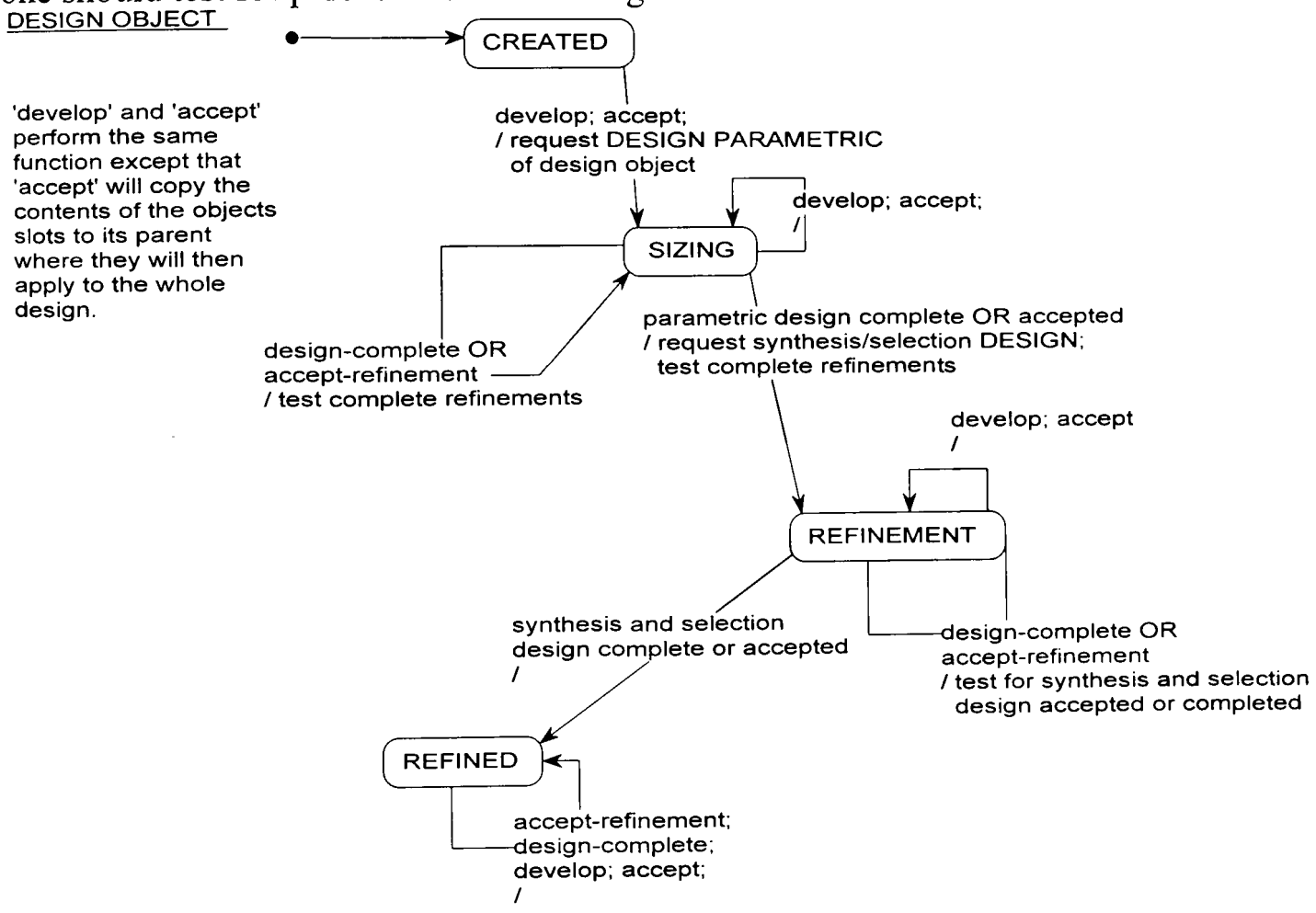
Name: set-value-for-parent [DESIGN OBJECT.]
Type: Operation
Description:
Objects may share slots with its parents if the object has been selected or involved in a parametric design process. Therefore if a value is set, then it has to be assigned to the object where the slot is ensured to be consistent throughout its design (i.e. where the slot is first defined in a type hierarchy). If a value is null in an object it does not mean there is no value assignment, if the slot also belongs to a parent then the parent may store the value. 'set-value-for-parent' works in conjunction with get.

If the slot has dependencies and the value has been set, then the dependencies to that slot must be deleted as well. If the design has been accepted, then the 'cost-of-change', i.e. the effect of changing the slot, should have been accounted for.

A check is made if the parent-design-object is null in the case where the object is the first in the hierarchy, i.e. the origin object.

Name: synthesise [DESIGN OBJECT.]
Type: Operation
Description:
Inform the design object that it has been synthesised to a set of specified design objects (a proposal).

Name: test-parameter [DESIGN OBJECT.]
Type: Operation
Description:
Returns true if the slot has been assigned a value. This is valuable in a design process, as an agent needs to be able to test if the appropriate values are available to perform the design BEFORE recording any dependency with the slot. If the object was derived from a parametric design process, then the slot value may be high in the hierarchy, and therefore one should test for parent values not being null etc.



State transition Diagram for the Design Object

State: CREATED
Description:
When an object is created, its state remains in the created state until someone has issued a request that it be developed.

State: SIZING
Description:
Sizing mode is complete when all interested agents have responded to the design request.

When sizing is identified to be complete, this does not indicate that additional sizing functions can be performed. It is however more likely that design re-work will be required as other design agents will have based their proposals on the previously accepted parameters of the design object,

'parametric design is complete' will be true in this state if all agents have reviewed the object and do not present a proposal.

State: REFINEMENT
Description:
Indicates that sizing has been completed and that agents have been requested to refine (synthesise and select) the design object. A new parametric design for the object may appear in later design stages that may possibly invalidate the refinement, but because it cannot be determined when these cases will arise, it can be assumed that the design can proceed after each later parametric refinement.

State: REFINED
Description:
When the design object has been sized, and both the selection refinement and synthesis refinement has been completed, then the objects design is complete, from the point of view of its children. It is then its childrens responsibility to be designed.

DEPENDENTS

Maintains a list of proposals that are dependencies to a specific object slot. There is a DEPENDENTS object for each slot in the DESIGN_OBJECT.

Name: delete-flag [DEPENDENTS.]
Type: Attribute
Description:
Indicates whether object selected for delete. Default FALSE.

Name: dependencies [DEPENDENTS.]
Type: Attribute
Description:
A multislot of pointers to type proposal. These proposals used the design slot to justify the proposal itself, so if the value is changed/deleted the proposals have to be re-accessed.

Name: add [DEPENDENTS.]
Type: Operation
Description:
Add dependency to the dependency list. The dependency is only added if the dependency does not already exist in the list.

Name: add-list [DEPENDENTS.]
Type: Operation
Description:
Add a list of dependencies to the dependency list. Takes a multislot list of proposals.

Name: delete-dependents [DEPENDENTS.]
Type: Operation
Description:
Delete all proposals in the dependencies list. The dependencies list itself is not deleted.

Name: delete-except [DEPENDENTS.]
Type: Operation
Description:
Delete all proposals except the one passed by reference. This is used when 'set-value-for-parent' in the DO.

Name: get-dependency-list [DEPENDENTS.]
Type: Operation
Description:
Return a multislot list of proposals, the 'dependencies'.

SLOT

A slot is a part of the design object. The methods considered here for the slot are not developed in this module but are identified for ease of understanding.

Name: dependencies [SLOT.]
Type: Attribute
Description:
A slot can have a number of dependent proposals. This means that a slot has been used in preparation of a proposal. If the slot is modified, then the basis of the proposal in now invalid, the proposal has to be removed, and the situation re-accessed.

A slot has a null value by default when created. This is so that one can determine what values have been set by an agent making a proposal.

Name: parameter-value [SLOT.]
Type: Attribute
Description:
A parameter value can be of any type. It is null by default or if the value is deleted.

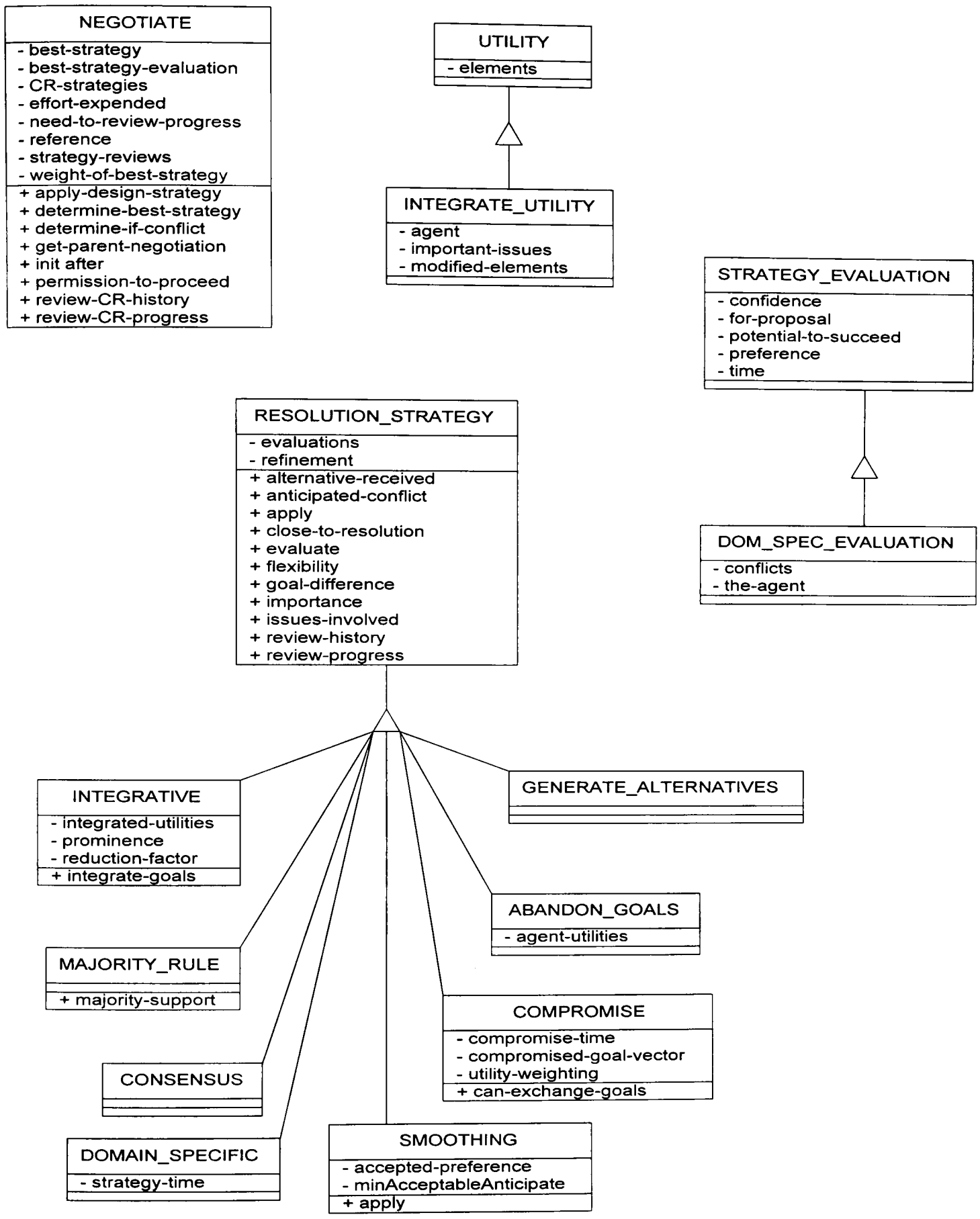
Name: get-parameter [SLOT.]
Type: Operation
Description:

Returns the value of the current slot, and makes a record of the design object currently under design that requires the slot value in order to justify its design proposal.

Name: set-parameter [SLOT.]
Type: Operation
Description:
Set the value of the specified slot. If the slot has dependencies then a conflict should be raised and the value of the slot should not be set. Logically it should only change a value by raising a conflict and rejecting it. An alternative to the current design can then be proposed.

Name: test-parameter [SLOT.]
Type: Operation
Description:
Returns true if the slot has been assigned a value. This is valuable in a design process, as an agent needs to be able to test if the appropriate values are available to perform the design BEFORE recording any dependency with the slot.

Negotiation



NEGOTIATE

The negotiation object is created for a refinement in order to resolve conflict. Negotiation

objects cooperate in a network to resolve design problems and control the design process.

Name: best-strategy [NEGOTIATE.]
Type: Attribute
Description:
Value recorded by 'determine-best-strategy' when a best strategy can be found. Contains address of RESOLUTION_STRATEGY.

Name: best-strategy-evaluation [NEGOTIATE.]
Type: Attribute
Description:
Value recorded by 'determine-best-strategy' when best strategy was found. Contains address of STRATEGY_EVALUATION.

Name: CR-strategies [NEGOTIATE.]
Type: Attribute
Description:
A list of the conflict resolution strategies that can be applied.

Name: effort-expended [NEGOTIATE.]
Type: Attribute
Description:
Denotes the time spent on this particular design path. It is a summation of all the validated (see 'permission to proceed') requests for design to proceed. It is a fuzzy value which is a summation of the effort specified as required by the conflict resolution mechanisms.

Name: need-to-review-progress [NEGOTIATE.]
Type: Attribute
Description:
This is set to TRUE by a negotiation strategy if there is a requirement for the strategy to review progress and give permission for further design to proceed.

Name: reference [NEGOTIATE.]
Type: Attribute
Description:
The reference to the refinement object associated with the negotiation object.

Name: strategy-reviews [NEGOTIATE.]
Type: Attribute
Description:
A list of strategy reviews that were generated by method determine-best-strategy. The list of strategies with their corresponding STRATEGY_EVALUATIONS. This list is only generated if their was conflict.

Name: weight-of-best-strategy [NEGOTIATE.]
Type: Attribute
Description:
When the best strategy is determined, the weight given to this best strategy is recorded here. This is used to determine if the design is improving or sacrifices getting larger (see permission-to-proceed).

Name: apply-design-strategy [NEGOTIATE.]
Type: Operation
Description:
The best strategy (denoted by slot best-strategy) is requested to 'apply' the best-strategy-evaluation.

Name: determine-best-strategy [NEGOTIATE.]
Type: Operation
Description:
Determines which conflict resolution strategy is the most appropriate. Calls evaluate method for each of the resolution strategies and passes the REFINEMENT pointer.

The evaluate method returns a set of STRATEGY_EVALUATIONS. The best one is determined by applying a utility function, (utility weightings specified by the user in instance 'user_CR_criteria') and the best one selected.

The strategies are recorded in the slot strategy-reviews.

Returns TRUE if best strategy was found, FALSE if no strategy. If TRUE, then the best strategy and evaluation are recorded in 'best-strategy' and 'best-strategy-evaluation' accordingly.

Name: determine-if-conflict [NEGOTIATE.]
Type: Operation
Description:
Determine if a conflict exists in a proposal.

A conflict exists if: - agents confidence in a particular solution is low (even though it is it's best guess. This would be represented by a soft conflict rather than by an evaluation. - agents disagree on the best proposal - there are conflicts associated with the best proposal.

The method returns the best proposal if no conflict exists, or nil otherwise.

Name: get-parent-negotiation [NEGOTIATE.]
Type: Operation
Description:
Returns the address of the parent negotiation object, or nil if there is no parent negotiation

object (i.e. the ORIGIN).

Name: init after [NEGOTIATE.]
Type: Operation
Description:
Create the conflict resolution strategies for this particular refinement. This is important so that the conflict resolution objects can maintain review and history information for this particular refinement.

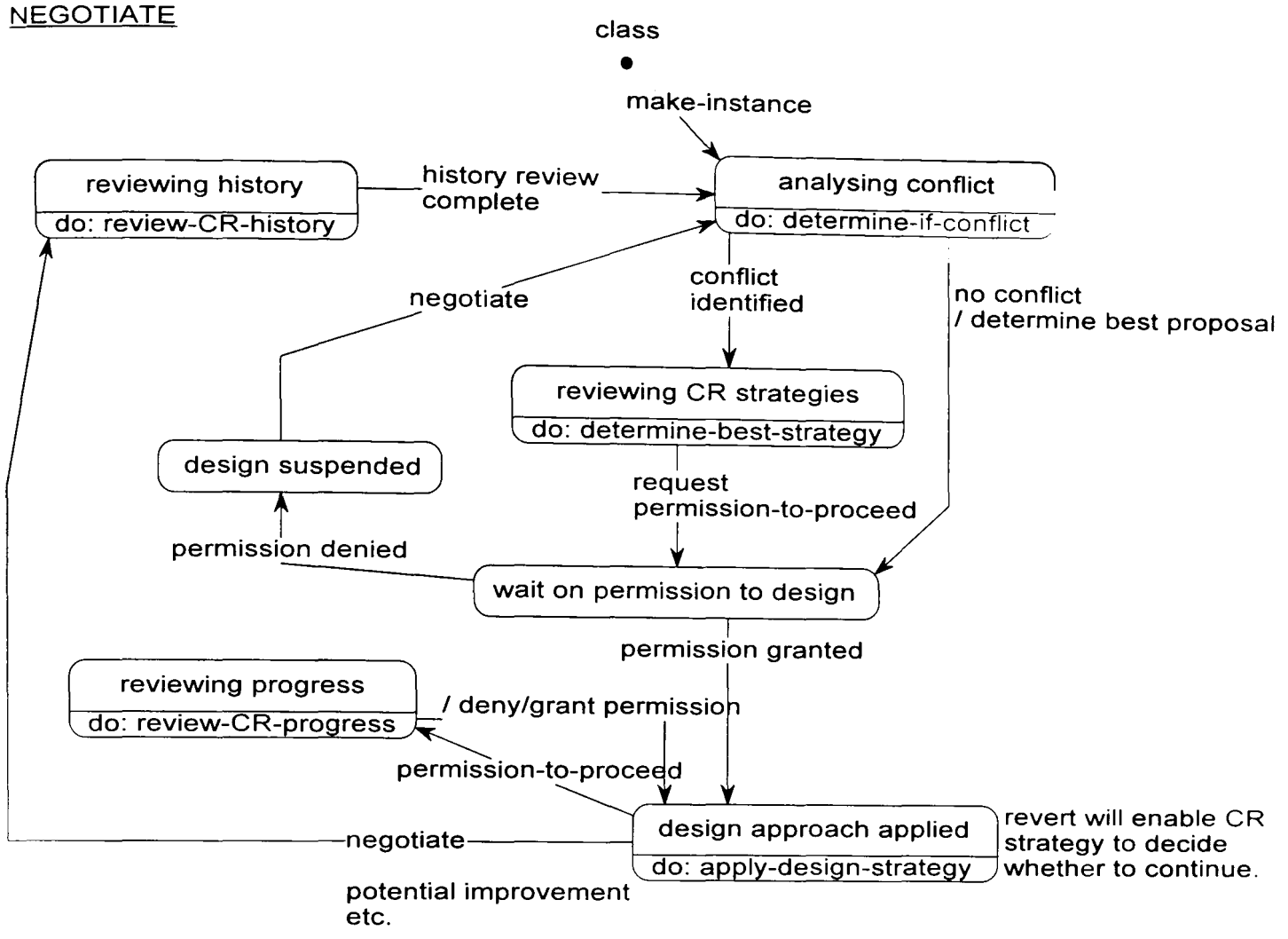
Name: permission-to-proceed [NEGOTIATE.]
Type: Operation
Description:
'permission-to-proceed' is called by another negotiation mechanism involved further down the design chain when a conflict is identified. It is a mechanism through which if the design is not progressing as successfully as expected, the negotiation mechanism can halt that particular design path and select other alternatives.

The selected STRATEGY_EVALUATION is passed as an argument. If accepted, the time is recorded within the current object, as it can be counted for as 'time applied' in attempting to resolve conflict in this refinement.

The 'strategy-reviews' list is temporarily modified as follows: the best-strategy-evaluation is removed from the list; the strategy passed as an argument is added to the list. The best strategy is then selected from the list. If the best strategy is the one passed as an argument then permission to proceed can be granted, otherwise denied.

Name: review-CR-history [NEGOTIATE.]
Type: Operation
Description:
This message is sent when problems still exist in the design. It is useful where the conflict resolution mechanism wishes to review progress as a result of applying its strategy.

Name: review-CR-progress [NEGOTIATE.]
Type: Operation
Description:
The design approach has been applied, and the later design stage has requested for design to continue. Review the strategy and determine whether design should proceed. If it is determined that the strategy is not well applied, then return false. and permission to proceed is denied, otherwise true is returned.



State Transition Diagram for the Negotiation Object

State: ANALYSING CONFLICT
Description:
Determine if a conflict exists.

State: REVIEWING CR STRATEGIES (Conflict Resolution Strategies)
Description:
Review strategies to determine best approach.

State: WAIT ON PERMISSION TO DESIGN
Description:
With best solution to problem, request permission if acceptable and request further design.

Informs the parent negotiation object that either design should not continue along this path, or a request to perform a design when the benefits do not appear as great as initially expected. Requests permission-to-proceed and determines if permission is required.

State: DESIGN APPROACH APPLIED
Description:
The selected approach has been applied and as yet there is no feedback.

State: REVIEWING PROGRESS
Description:
If a specific CR strategy is interested in following progress, for example, bring more information into the CR process, then we review progress when a revert is made.

State: DESIGN SUSPENDED
Description:
Design has been suspended for the time being while another design route is accessed.

Obviously the events deny/grant permission do not happen in this state, as there should be no design occurring further down the particular design path.

State: REVIEWING HISTORY
Description:
A conflict has previously been identified and a strategy applied. The conflict resolution strategy in some cases may wish to review the progress of the design as a result of conflict resolution. The strategy has a chance to do this when the agents have put forward new proposals as a result of the conflict resolution results.

RESOLUTION_STRATEGY

A conflict resolution strategy is able to determine if a conflict can be resolved, resolve the conflict, and to determine the success of the strategy. All specific resolution strategies inherit that functionality from this base RESOLUTION_STRATEGY.

Name: evaluations [RESOLUTION_STRATEGY.]
Type: Attribute
Description:
A list of STRATEGY_EVALUATIONS put forward by strategy. See STRATEGY_EVALUATIONS.

Name: refinement [RESOLUTION_STRATEGY.]
Type: Attribute
Description:
A reference to the refinement which the strategy is analysing. The value is set by the evaluate method.

Name: alternative-received [RESOLUTION_STRATEGY.]
Type: Operation
Description:
This method is called when an alternative has been received after a request for an ALTERNATIVE has been requested by the negotiation mechanism. Note that the proposal received has not been evaluated, this is the responsibility of the negotiation strategy. The

method takes the proposal provided as an alternative, and the old proposal respectively.

Name: anticipated-conflict [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Determines the degree of anticipated conflict for a proposal.

Anticipated conflict is determined from analysis of the extent of conflict, and the belief in the conflict. If the degree of belief in a conflict is high, then smoothing is not appropriate as more information is less likely to find a situation where the conflict is not apparent.

low belief + low preference = high anticipated conflict

(1-belief) * (1-preference) = degree of anticipated conflict

Name: apply [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Apply the strategy to the problem. A STRATEGY_EVALUATION is passed as argument. This evaluation was the one proposed by the resolution strategy that has been accepted.

Name: close-to-resolution [RESOLUTION_STRATEGY.]
Type: Operation
Description:
How close the parties are to resolving the conflict for a specified proposal.

Returns 0 if there is a hard conflict. Otherwise returns the largest degree of difference between the evaluations and the proposal in question.

Name: evaluate [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Determine if the strategy can be applied successfully in this situation. Returns a list of evaluations of the type STRATEGY_EVALUATION (potentially for each proposal in the refinement). Takes the REFINEMENT as the argument.

Name: flexibility [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Determines the flexibility of a proposal. Flexibility is determined by the absence of hard conflicts, and degree of prominence. A higher degree of prominence indicates that an alternative solution that can be provided by the agent is less likely to be acceptable.

Takes a proposal as an argument. Returns a factor between 0 and 1, 1 being highly

flexible.

Name: goal-difference [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Determine the degree of difference between a set of goals.

The goal difference is determined as follows: For all the different utility vectors applied, the difference between the best and worst case for each issue is determined. The differences for each issue are summed, and this total divided by the number of utility vectors considered. This result will be within the range 0-1, as the worst case difference for any number of issues will be equal to the number of utility vectors considered.

Name: importance [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Determines the highest maximum-preference factor for a proposal and its set of evaluations/conflicts.

Name: issues-involved [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Returns a vector the length of the normal evaluation vector (issues). A '1' in a vector position indicates that at least one of the agents who either presented the proposal, conflict or evaluation considers that the associated value is important (i.e. not nil).

Name: review-history [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Review history of conflict resolution. Takes the strategy evaluation as an argument so it knows the strategy that was applied.

Name: review-progress [RESOLUTION_STRATEGY.]
Type: Operation
Description:
Takes object of STRATEGY_EVALUATION as argument which for information on how well progress is going. Returns FALSE if permission to continue design is denied, TRUE otherwise.

INTEGRATIVE

An integrative strategy is a search for the 'win-win' situation, where both parties collaborate to develop a solution that matches both their needs. It is essentially a trade in values,

where something of little importance to one party may be of considerable importance to another.

Strategy works by: - identifying the issues most important to each agent reviewing the proposal. - determine the most important issues from all viewpoints. - Integrate the agent utilities by reducing the value of those utilities that are considered not to be important from the global perspective, and increase the important values.

Name: integrated-utilities [INTEGRATIVE.]
Type: Attribute
Description:
A list of utilities by agent that have been integrated. The slot holds a list of type INTEGRATE_UTILITY.

Name: prominence [INTEGRATIVE.]
Type: Attribute
Description:
Denotes the degree of difference between an issue and the smallest specified issue whereby the higher issue is regarded as critical and therefore can be exchanged. Variable used by the method integrate-goals.

Name: reduction-factor [INTEGRATIVE.]
Type: Attribute
Description:
Denotes the degree of reduction of an issue if it is not important to the global perspective.

Name: integrate-goals [INTEGRATIVE.]
Type: Operation
Description:
Takes a list of proposals over which to integrate the goals, and records a list of agents with corresponding utilities in the slot integrated-utilities.

MAJORITY_RULE

Majority rule is where the group will vote on the solution. If there are more conflict than assessments then there is not majority support.

- majority rule is appropriate where parties are not close to resolving conflict - the majority rule strategy is good where reaching agreement is imperative or it does not matter at all

Name: majority-support [MAJORITY_RULE.]
Type: Operation
Description:

Returns true if most agents support a proposal (i.e. no conflicts - either hard or soft). Simple function that assumes agent only puts forward a single proposal in assessment.

CONSENSUS

Consensus is where the parties work together to find the best solution by bringing harmony among the conflicting requirements.

Consensus will accept a proposal where there is general agreement from all agents that the proposal is ok. It might not be the best from the viewpoint of any individual agent.

- if hard constraint then a consensus strategy is not appropriate - appropriate where the value of reaching agreement is of a low importance

DOMAIN_SPECIFIC

Domain specific conflict resolution strategy. This strategy attempts to determine if another agent can provide a solution that resolves the identified conflict.

Name: strategy-time [DOMAIN_SPECIFIC.]
Type: Attribute
Description:
The time expected to be taken in applying the strategy. This value is low as specific resolution strategies are expected to resolve the conflict. This is a fuzzy factor between 0-1, 1 being user involvement (which is time expensive), and 0 being no time at all.

GENERATE_ALTERNATIVES

The search for alternative solutions that can satisfy all parties. The tradeoff between the cost of search and changing the individuals ideals is addressed by the negotiation strategy.

ABANDON_GOALS

Where the less important goals from the parties involved in conflict are dropped in order to resolve conflict, but their main concerns are accounted for.

Name: agent-utilities [ABANDON_GOALS.]
Type: Attribute
Description:
A list of {<agent><utility>}. The utility represents the goals applied previously in the evaluation of the proposal.

COMPROMISE

A compromise is made by applying 'middle values' that are important to each of the agents involved in the conflict.

Name: compromise-time [COMPROMISE.]
Type: Attribute
Description:
A quantitative assessment of a time required to complete a compromise resolution strategy.

Name: compromised-goal-vector [COMPROMISE.]
Type: Attribute
Description:
A vector with the compromised goal derived in the last call to apply. This vector is recorded so that evaluations with the compromised vector can be performed when the alternative (compromised) design solution is received.

Name: utility-weighting [COMPROMISE.]
Type: Attribute
Description:
A ranking of the importance of the issues in the following order: close to resolution, single issue, agents flexibility, importance of reaching agreement.

Name: can-exchange-goals [COMPROMISE.]
Type: Operation
Description:
Determines whether all the agents involved with a particular proposal can exchange goals. Takes the proposal as an argument.

SMOOTHING

Where a conflict is due to a lack of information, a smoothing strategy can be applied to bring new information into the conflict in order to resolve it. If for example the confidence in a solution is low, one can progress design a little further to determine is the confidence improves. The smoothing strategy works by getting the agents to develop the solution further - without actually accepting the proposal.

- Only put forward proposal for smoothing strategy if degree of anticipation is greater than a minimum (minAcceptableAnticipate)

Name: accepted-preference [SMOOTHING.]
Type: Attribute

Description:
When the strategy is selected, the preference of the proposal is recorded in this attribute. This is necessary for comparison with more detailed design to determine if design is progressing.

Name: minAcceptableAnticipate [SMOOTHING.]
Type: Attribute
Description:
Minimum level of anticipation for which a proposal for the smoothing strategy will be presented to the negotiation mechanism.

Name: apply [SMOOTHING.]
Type: Operation
Description:
Informs the proposal in the selected strategy evaluation to develop itself. This will call appropriate design functions in each of the design objects in the proposal. Develop it used to test a design approach if confidence is low.

UTILITY

Holds a utility vector.

Name: elements [UTILITY.]
Type: Attribute
Description:
The utility vector as a multi-slot (array).

INTEGRATE_UTILITY

A utility object that represents additional information required by the INTEGRATE conflict resolution strategy.

Name: agent [INTEGRATE_UTILITY.]
Type: Attribute
Description:
The reference to the agent to which the goal structure applies.

Name: important-issues [INTEGRATE_UTILITY.]
Type: Attribute
Description:
A vector of the most important issues determined by the INTEGRATIVE resolution strategy, 1 indicating important, 0 otherwise.

Name: modified-elements [INTEGRATE_UTILITY.]
Type: Attribute
Description:
A record of the modified utilities of an agent as determined by the INTEGRATIVE conflict resolution mechanism.

STRATEGY_EVALUATION

An instance of this is returned from the method 'evaluate' in object RESOLUTION_STRATEGY. It identifies the appropriateness if the strategy in resolving the conflict.

The perfect values for the attributes are: confidence high potential to succeed high preferencehigh time low

Time is therefore inverted (1-n) by utility function.

Name: confidence [STRATEGY_EVALUATION.]
Type: Attribute
Description:
The 'confidence' from the best proposal which the mechanism considers conflict can be resolved. Value in range 0-1.

Name: for-proposal [STRATEGY_EVALUATION.]
Type: Attribute
Description:
A reference to the proposal that the evaluation is for.

Name: potential-to-succeed [STRATEGY_EVALUATION.]
Type: Attribute
Description:
The potential of the CR mechanism to resolve the conflict. Value in range 0-1. If 0, conflict cannot be resolved.

Name: preference [STRATEGY_EVALUATION.]
Type: Attribute
Description:
The 'preference' from the proposal that the CR mechanism considers conflict can be resolved. Value in range 0-1.

Name: time [STRATEGY_EVALUATION.]
Type: Attribute
Description:

Time required by CR mechanism. Value in range 0-1.

Time is not recorded 'as is' but as a degree between the following qualitative factors: 0.02 single shot 0.05 low iterative 0.3 high iterative 0.8 potentially infinite 1.0 user involvement

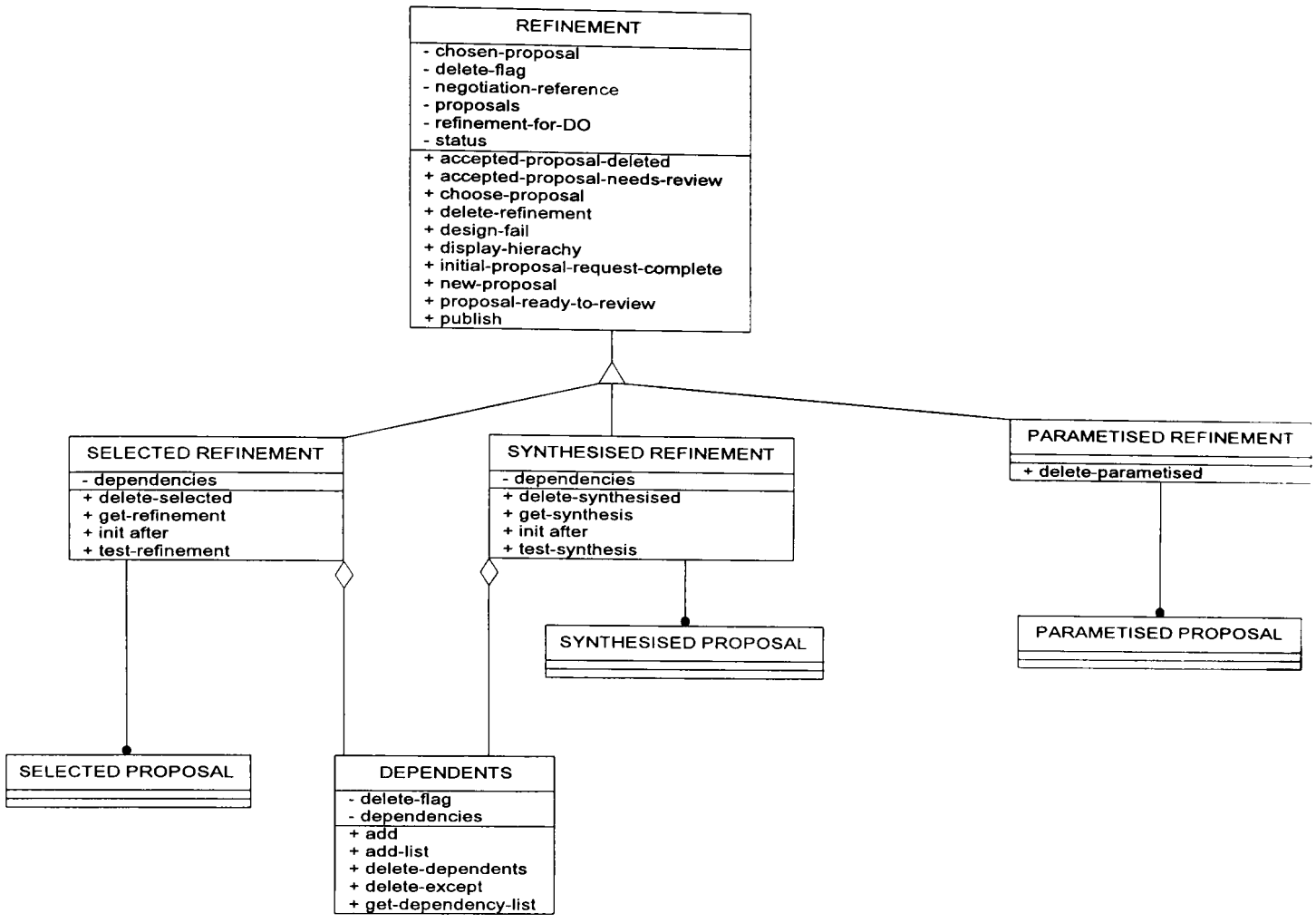
DOM_SPEC_EVALUATION

Strategy evaluation used by the domain specific strategy. It records additional information pertinent to the application of the strategy.

Name: conflicts [DOM_SPEC_EVALUATION.]
Type: Attribute
Description:
A list of references to the conflict objects which can be resolved by the specified agent.

Name: the-agent [DOM_SPEC_EVALUATION.]
Type: Attribute
Description:
The reference to the agent who can resolve the conflicts.

Refinement



REFINEMENT

A REFINEMENT is a chosen design for the particular design object. Only one refinement proposal can ultimately be chosen. A REFINEMENT however keeps track of the many proposals suggested for the refinement and requests the services of the negotiation mechanism to resolve conflict.

Name: chosen-proposal [REFINEMENT.]
Type: Attribute
Description:
The proposal that was chosen as the best 'design' for a particular part (parametric, synthesis, selection) of the design object. The chosen-proposal is null by default.

Name: delete-flag [REFINEMENT.]
Type: Attribute
Description:
Indicates if refinement has been deleted. Default FALSE.

Name: negotiation-reference [REFINEMENT.]
Type: Attribute
Description:
A reference to the negotiation object. Default nil. Value set when negotiation object is

created.

Name: proposals [REFINEMENT.]
Type: Attribute
Description:
A list of proposals for the particular refinement.

Name: refinement-for-DO [REFINEMENT.]
Type: Attribute
Description:
Records the design object for which the REFINEMENT is attached.

Name: status [REFINEMENT.]
Type: Attribute
Description:
Status of a refinement can be: REQUIRE_DESIGN, DESIGN_UNDER_REVIEW,
DESIGN_ACCEPTED, DESIGN_COMPLETE,
POTENTIAL_DESIGN_IMPROVEMENT,
POTENTIAL_DESIGN_ERROR.

Name: accepted-proposal-deleted [REFINEMENT.]
Type: Operation
Description:
A message to the refinement from the proposal that the accepted proposal has been deleted.

Name: accepted-proposal-needs-review [REFINEMENT.]
Type: Operation
Description:
A message to the refinement to indicate that the accepted proposal needs review (and therefore is more critical to review as their is a potential design error).

Name: choose-proposal [REFINEMENT.]
Type: Operation
Description:
A proposal in the proposals list is selected for the design refinement. If another proposal was previously selected, then any dependencies on that proposal must be deleted. The method records the 'chosen-proposal'.

Name: delete-refinement [REFINEMENT.]
Type: Operation
Description:
Delete proposals and the refinement itself. No check is made to ensure that the delete-flag is set, this is accounted for in the sub-classes.

Name: design-fail [REFINEMENT.]
Type: Operation
Description:
The negotiation mechanism could not decide on a solution for the refinement, therefore it sends the message design-fail. Nothing is requested from the environment here (agents) but agents can propose something in the future which would be considered.

Name: display-hierarchy [REFINEMENT.]
Type: Operation
Description:
Display information concerning this synthesis, and the proposals of this synthesis. Each proposal is requested to display its details. An accepted proposal (if their is one) is requested to display details of all its children (further design). This operation can only be called from the design object (which maintains the logical router).

Name: initial-proposal-request-complete [REFINEMENT.]
Type: Operation
Description:
Indicates to the REFINEMENT that no more proposals for the initial design request are to be presented. This does not prevent further design proposals being received - but they are done on the back of the negotiation mechanism, or presented in an -ad hoc fashion by the agent.

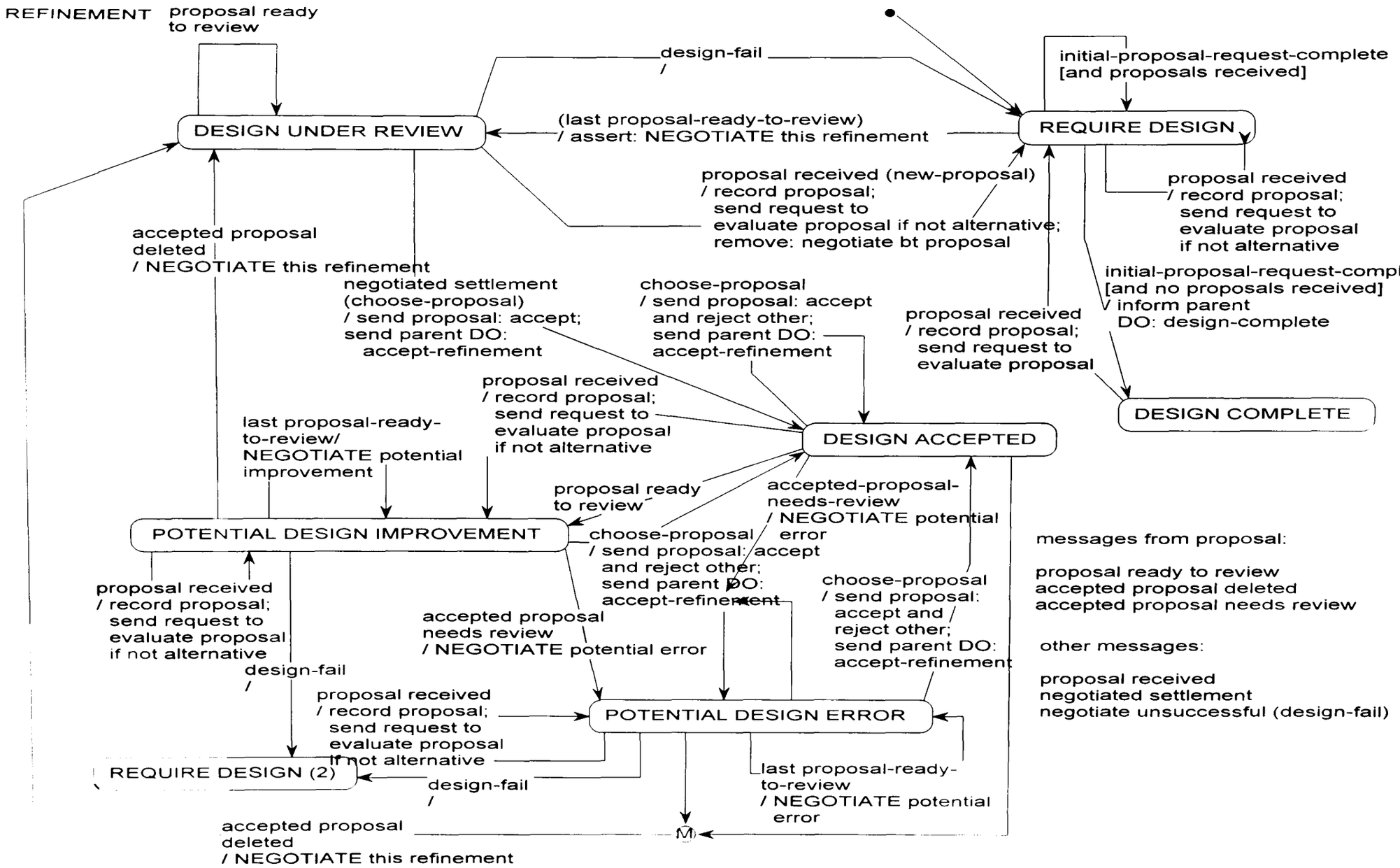
Name: new-proposal [REFINEMENT.]
Type: Operation
Description:
A new proposal is added to the refinement for consideration.

Inform the proposal that it is open-to-review

Name: proposal-ready-to-review [REFINEMENT.]
Type: Operation
Description:
This message is sent to the refinement when one of the proposals in the refinement is ready for review. The REFINEMENT checks to determine if all the proposals are ready to review in order to determine if the refinement can be reviewed.

Name: publish [REFINEMENT.]
Type: Operation
Description:
Display information concerning the refinement and request additional information from the accepted proposal.

State Transition Diagram for the Refinement Object


REFINEMENT
proposal ready to review
DESIGN UNDER REVIEW
design-fail /
initial-proposal-request-complete [and proposals received]
REQUIRE DESIGN
(last proposal-ready-to-review) / assert: NEGOTIATE this refinement
proposal received (new-proposal) / record proposal; send request to evaluate proposal if not alternative; remove: negotiate bt proposal
proposal received / record proposal; send request to evaluate proposal if not alternative
accepted proposal deleted / NEGOTIATE this refinement
negotiated settlement (choose-proposal) / send proposal: accept; send parent DO: accept-refinement
choose-proposal / send proposal: accept and reject other; send parent DO: accept-refinement
initial-proposal-request-complete [and no proposals received] / inform parent DO: design-complete
proposal received / record proposal; send request to evaluate proposal
DESIGN COMPLETE
proposal received / record proposal; send request to evaluate proposal if not alternative
last proposal-ready-to-review/ NEGOTIATE potential improvement
DESIGN ACCEPTED
proposal ready to review
accepted-proposal-needs-review / NEGOTIATE potential error
POTENTIAL DESIGN IMPROVEMENT
choose-proposal / send proposal: accept and reject other; send parent DO: accept-refinement
messages from proposal:
proposal ready to review
accepted proposal deleted
accepted proposal needs review
other messages:
proposal received
negotiated settlement
negotiate unsuccessful (design-fail)
proposal received / record proposal; send request to evaluate proposal if not alternative
accepted proposal needs review / NEGOTIATE potential error
choose-proposal / send proposal: accept and / reject other; send parent DO: accept-refinement
design-fail /
proposal received / record proposal; send request to evaluate proposal if not alternative
POTENTIAL DESIGN ERROR
REQUIRE DESIGN (2)
design-fail /
last proposal-ready-to-review / NEGOTIATE potential error
accepted proposal deleted / NEGOTIATE this refinement
M

REFINEMENT

State: REQUIRE DESIGN
Description:
The refinement has been created for a particular design object and is ready to receive design proposals.

State: DESIGN COMPLETE
Description:
The design is complete when no proposals have been received for a particular design requirement.

This is critical in parametric design where we wish to know that the parametric design can not proceed further and therefore signal the parent object that initiated the parametric design process that other design processes can continue. i.e. it is best to wait for parametric design to complete before proceeding with the selection and synthesis design processes.

State: DESIGN ACCEPTED
Description:
All interested agents have accepted a single proposal put forward for the design synthesis of the design object in question. There are no proposals (or evaluations) that have not been reviewed in this state.

State: POTENTIAL DESIGN ERROR
Description:
An accepted proposal has been reviewed and there is additional information (evaluations etc) available that was not accounted for when the negotiation took place. There could therefore be a potential design error that must be managed.

State: POTENTIAL DESIGN IMPROVEMENT
Description:
A design proposal has been previously accepted, although another proposal has been put forward and reviewed that obviously has the potential to be better than the proposal currently accepted as part of the design. This state signifies the fact that a potential design improvement is available.

State: DESIGN UNDER REVIEW
Description:
The design is under review by the negotiation mechanism.

If the design fails, i.e. the negotiation mechanism cannot determine a good solution (without conflict), then the refinement becomes REQUIRE DESIGN, and no other evaluation effort is expended unless further proposals are put forward at a later date.

SELECTED REFINEMENT

A refinement is essentially a pointer to the chosen proposal that was selected as a good selection of the current design object. Proposals for review are also recorded here.

The test is useful for agents to analyse plant structure.

Agents will propose possible selections (or refinements) for a particular design object. There can be many proposals. A proposal for a selection will contain a single object that has to be a sub-class of the object being selected.

Name: dependencies [SELECTED REFINEMENT.]
Type: Attribute
Description:
A list of the dependencies associated with the current design refinement. The dependencies are proposals.

Name: delete-selected [SELECTED REFINEMENT.]
Type: Operation
Description:
Deletes dependent proposals (recorded in dependencies) and calls delete-refinement. Does check to avoid loops in deletion.

Name: get-refinement [SELECTED REFINEMENT.]
Type: Operation
Description:
Get the design object name and type. The proposal being prepared that requires this information is recorded as a dependency. The chosen-proposal is returned.

Name: init after [SELECTED REFINEMENT.]
Type: Operation
Description:
Initialise the dependents list

Name: test-refinement [SELECTED REFINEMENT.]
Type: Operation
Description:
Tests whether an object has been refined or not. Useful when a design rule in an agent cannot proceed without this type of knowledge and therefore does not wish to record a dependency.

SYNTHESISED REFINEMENT

Manages the synthesis proposals for the associated design object. The refinement maintains a list of proposals containing design objects. An agent will propose a set of design objects that satisfy the requirements of the parent design object. There can be many proposals for a particular design object.

Dependencies are recorded as an agent may which to review plant structures before presenting proposals.

Name: dependencies [SYNTHESISED REFINEMENT.]
Type: Attribute
Description:
Dependent proposals are recorded that are based on information regarding plant structure.

Name: delete-synthesised [SYNTHESISED REFINEMENT.]
Type: Operation
Description:
Delete synthesis refinement. Deletes dependencies then calls its delete-refinement function. Performs a check to ensure that the refinement is not already part of the delete chain.

Name: get-synthesis [SYNTHESISED REFINEMENT.]
Type: Operation
Description:
Get a list of design objects that the current design object has been synthesised to. Record the dependency to the proposal that is currently being developed that requires this information in taking its decisions. It is important to review this synthesis rather than going to view the children (proposals).

Name: init after [SYNTHESISED REFINEMENT.]
Type: Operation
Description:
Create DEPENDENTS list for dependencies.

Name: test-synthesis [SYNTHESISED REFINEMENT.]
Type: Operation
Description:
Tests if an object has been synthesised (not whether any synthesis proposals exist). This is useful where the design rule in an agent cannot proceed without this information and therefore does not wish to record a dependency to this information at this stage.

PARAMETISED REFINEMENT

The type of REFINEMENT that records parametric proposals for a design object. See REFINEMENT for details.

Note that there is no dependency to other design objects. No agents are interested if the object has been parametised or not. Agents are interested in the parameters (slots) and they keep their own dependency links.

Name: delete-parametised [PARAMETISED REFINEMENT.]
Type: Operation
Description:
Delete parametric refinement. Since the refinement has no dependencies itself, it calls its own delete function. Does a check to ensure no already in a delete list.

Appendix L. Detailed Grammar Specification

Dictionary

The language definition conforms to the BNF notation.

<agent>	The name of the agent.
<design-method>	The design method, either 'selection', 'parametric', or 'synthesis'. It can be 'null' (by default) if the proposal is a request for the design process to proceed, e.g. the user wants a design for a heat transfer system from an agent (or user).
<req>	A textual description of a requirement. Must hold a text description that can be given to an engineering user - a reason why for example an alternative solution must be provided. This is to allow normal human interaction with the system.
<reason>	The reason for a particular conflict. A reason can be either formal or informal. Informal: a description (textual) of why a conflict existed Formal: a notation enabling an agent to understand and apply a relevant conflict resolution strategy.
<obj> <objReq>	A reference to a design object.
<importance>	The importance ascribed to particular problem. It can denote whether a conflict is due to a hard or soft constraint violation, and identify a particular dislike to a solution.

PROPOSE

Syntax: **PROPOSE** <proposalObject> [[**for** <desObj>] | [**for** <proposal>]] **from** <agent> [**as** [**initial** | **alternative** <altern-to-proposal>]]

Parameters:

<proposalObject>	[<parametised proposal> | <synthesised proposal> | <selected proposal> | <conflict proposal> | <evaluation proposal>]

<desObj>	The object to which the proposal is associated with (e.g. A design such as a refinement).
<proposal>	The proposal to which the proposal should be associated with. This is specified for evaluations and conflicts which are associated with option proposals rather than particular objects.
<agent>	agent presenting the proposal
[initial | alternative]	indicates whether the proposal was a result of a design request (initial) or an alternative (after a request to provide an alternative). The specification of initial or alternative only makes sense for an OPTION proposal.
<altern-to-proposal>	A reference to a proposal. If the proposal is an alternative to one previously presented, then the alternative proposal must be specified.

General notes:

If the <desObj> is not specified, it is assumed that it is new object that you wish the system to design the system or object.

Functionality:

For an OPTION PROPOSAL:

a. record keeping:

Copies the agent reference in the command to the proposal object.
Record <desObj> in derived-from-object in the proposal (record in proposal the object it was derived for).

b. if a design object is specified, then

if proposalObject is parametised, then 'parametise <proposalObject>' sent to <desObj>
elsif proposalObject is selected, then 'select <proposalObject>' sent to <desObj>
else proposalObject is synthesised, and 'synthesise <proposalObject>' sent to <desObj>

if the proposal is not an ALTERNATIVE
A RECEIVED message is asserted to indicate that the proposal was successfuly received. This enables WAIT statements to monitor that all

proposals have been received.
endif

c. send EVALUATE messages to capable review agents.

Note that if an object has higher level classes in the class hierachy then there may be rules which need to be evaluated to validate the proposal from a number of points of view (e.g. centrifugal, pump, equipment etc). These viewpoints only need to be evaluated if the proposal has had some of its attributes changed (or specified) by some design object lower in the hierachy.

d. if design object not specified then the object is a requirement to design something, therefore EVALUATE (see below).

However, first create a dummy origin object (unique to proposal), then assign the new proposal to the correct refinement (synthesis). This enables evaluations to be generated for the object.

e. in all the above cases

Both evaluations and conflicts can be considered proposals as they are just different points to consider (albeit regarding an object), and they are dependent (most probably) on other slots - thereby implicating the conflicts and evaluations into the arena of consistency maintenance and recording of dependencies.

Q. what happens when a proposal is deleted ?

For a CONFLICT PROPOSAL:

For a conflict proposal, we are not interested in everyone evaluating the conflict. If it is neccessary then agents may be requested to supply alternatives, make concessions, or apply domain specific conflict resolution strategies.

Conflicts occur :

- whenever agents wish to make them. If you consider an agent to be an engineering user, we will have to enable him to disregard or find faults at any time in the design process. A conflict can be identified late in the design process when an object has already been agreed. This is why we cannot wait for agreement to proceed based on the principle that 'no more conflicts will be identified'.

- only for proposals. Conflicts are collected for for proposals and considered when proposals reviewed.

Conflicts get priority on the agenda over evaluation proposals, as they can occur in an ad-hoc manner, and the node is ready for assessment when the last evaluation is received - we do not know when conflicts will arise.

a. copies the agent reference in the command to the proposal object (the conflict object).
b. get the parent-proposal from the specified design object. add-conflict <conflict-proposal> to the parent-proposal.

Q. what happens when a conflict is identified for a design that has already been agreed and refined ?

For an EVALUATION PROPOSAL:

An agent will be requested to evaluate a particular proposal if it has the capability. An evaluation proposal will be returned by the agent noting his level of agreement with the specified proposal.

Evaluations occur:

- after an agent has been requested to EVALUATE a specific proposal.
- whenever an agent feels it neccessary to propose an evaluation on part of the design.

After an agent has been requested to EVALUATE a design, we will enable design to procede with some confidence when the evaluations have been received from all those agents requested to participate (from knowledge of their interests). It is possible however, that an agent at a later stage may propose an EVALUATION for a design even though it was not requested to do so. This may be because an engineering user, or some other body, has decided that his initial guess or weighting of a proposal (confidence etc) may have been incorrect.

a. copies the agent reference in the command to the proposal object.

b. calls add-evaluation for the proposal (passed with the EVALUATE request) and passes the evaluation over to the proposal. The proposal is the parent-proposal to the specified design object passed in the request to evaluate.

Q. what is the difference between an evaluation and a conflict ?

A. An evaluation is a 'single shot' evaluation of a design that is summed up in a breif quantitative fashion. On the other hand an agent may present any number of conflicts for a particular proposal which identify the specific problems that pertain to the proposal to which the conflict is attached. A conflict is therefore more

'knowledge rich' compared to the evaluation.

DEVELOPE

Syntax: DEVELOP <desObj>

This is a request to a design object to design itself. When the negotiation mechanism wishes to inform an object to continue its design for review purposes, or a proposal has been accepted, then the objects must be sized, synthesised, and possibly go through a selection process. DEVELOP is a keyword that informs an object that these processes can begin.

DESIGN

Syntax: DESIGN <method> <desObj>

The design keyword is a request for the agents to design an object, i.e. to present their proposals.

The DESIGN request is issued when the design space has recieved all the appropriate evaluations that were requested for a proposed design and one has been selected and now requires further design.

The method is one of PARAMETRIC or SELECT_OR_SYNTHESISE. If parametric, then only objects that can perform a parametric design process can contribute.

ALTERNATIVE

Syntax: ALTERNATIVE <agent> required for <method> <obj> [because {<conflict>}] [using goals <utility-vector>]

An request from the negotiation mechanism to a specific agent to produce an alternative.

The agent responds to the request for providing an alternative by presenting a proposal. The agents previous proposal does not need to be retracted. The proposal will be evaluated (as with other proposals) and re-considered again as part of a group with all other proposals. Proposals will be retracted if two agents produce a compromise - as their previous proposals will still be their best proposals.

Q. How we we check that a request to provide an alternative has been evaluated and reply received ?
We do not. Only one alternative is requested at a time by the negotiation mechanism. Because there is only one, there does not need to be a wait state.

Q. How do we deteremine if an agent cannot provide an alternative because one does not exist ?
The agent presents an UNABLE_TO_PROPOSE message.

Q. If an agent provides an alternative but does not like to do so because he thinks that it is inadequate what does he do ?
A. Raise a conflict (soft constraint) noting his disagreement.

An alternative may be required because of a set of conflicts. The agent, if it has the capability (we do not assume that it does) can ensure that the next proposal generated does not disregard the conflicts that were identified with the initial proposal. These conflicts are parcelled as part of the request to provide an alternative ('because {<conflict>}').

If the agent can exchange goals, then he may be requested in a situation to apply certain goal criteria <utility-vector> in proposing his solution.

EVALUATE

Syntax: EVALUATE <proposal> for design of <desObj> [using goals <utility-vector>]

After a proposal is received it must be evaluated. An EVALUATION request is sent from the negotiation mechanism to the agent. The agent responds with a proposal of type evaluation. The design method is indicated by the type of proposal. An EVALUATION is not sent to the agent that put forward the proposal.

When the last EVALUATION is received, we will inform the parent of the proposal, that the last evaluation has been recieved.

The negotiation mechanism may request an agent to perform an analysis of a design proposal using certain goal criteria (for example - where the goals have been compromised). This new goal criteria to apply is specified in <utility-vector>.

REJECT

Syntax: REJECT <proposal>

An ACCEPTED message is sent from the negotiation space to the agent when a proposal is accepted. At the same time a REJECT message is sent to the appropriate agents for the remaining proposals. The reject message is sent to those agents that presented proposals and those that presented evaluations/conflicts for the proposal.

This is useful for management purposes if a system keeps track of its proposals and wishes to maintain a list of which ones have been accepted and which ones rejected. Note that a rejected proposal may become accepted at a later date due to the previously selected proposals causing design problems. Therefore if an agent keeps track of which proposals are reject and accepted, it would be wise not to delete a rejected proposal just because it had been rejected.

ACCEPTED

Syntax: ACCEPTED <proposal>

When a proposal is accepted, all agents that were involved in evaluating, proposing, and presented conflicts regarding the proposal will be informed. This is for management purposes if the agent wishes to keep track of the areas of design in which is was involved. The ACCEPTED command is sent from the negotiation mechanism to the agent when a proposal has been accepted. However, it is possible that problems later on in the design may cause the proposal to become rejected and therefore the agent may account for this.

FORCE_DELETE

Sytnax: FORCE_DELETE [<proposal> | <desObj>]

The DELETE message is sent from an agent to the negotiation layer and is considered to be a 'remove from consideration' command. Essentially it is the reverse of the proposal command with no requirement, i.e. an agent has requested an object to be designed (e.g. HTS). If the object has no parent, then there are no issues, we just have to delete all the dependencies. If the object however has a parent then the parent must be informed and the objects proposal deleted as well. Dependencies for the object must be removed.

Q. When an proposal is deleted and it has a parent refinement, do we select another of the parents proposals and continue design?
Q. If the proposal was the proposal that had been previously accepted (either unanimously by all cooperating agents or through the negotiation mechanism) then if deleted, the negotiation mechanism will be called in to evaluate the other

proposals (if any exist).

NEGOTIATE

Syntax: NEGOTIATE <refinement> because <reason>

This command is asserted by the REFINEMENT object when all the proposals become 'ready to review'. This fact indicates work for the negotiation mechanism which has to review the proposals, and conflicts (if any).

Parameters:

<reason> The reason why negotiation should procede. <reason> can be either:

'null'	for no reason, just negotiate a best proposal
'POTENTIAL_ERROR'	if there is a potential design error, i.e. accepted proposal has new conflict or review
'POTENTIAL_IMPROVEMENT'	if there is a potential design improvement, i.e. a new proposal has arrived in the refinement, or a new evaluation has arrived for another proposal in the refinement that was not accepted.

Syntax: NEGOTIATE <refinement> for <altern-proposal-object> as alternative to <old-proposal-object>

This message is used when alternative design proposals are recieved which have been requested by the negotiation mechanism.

Supplementary system messages:

AGENT_EVALUATE

Syntax: AGENT_EVALUATE <agent> to review <proposal> for design of <desObj> [using goals <utility-vector>]

Message to a specific agent to evaluate proposal.

AGENT_DESIGN

Syntax: AGENT_DESIGN <agent> to design <method> <desObj>

Message to agent to design (by <method>) the specified design object <desObj>

INTEREST

Syntax: *form1* INTEREST <agent> EVALUATE <design method> <object type>

form2 INTEREST <agent> DESIGN <design method> <object type>

Parameters:

<design method> either PARAMETRIC, SELECTION, or SYNTHESIS

form1 An <agent> asserts an interest in evaluating the particular <design method> design of <object type>. An agent is informed of events based on his interests. The objects he is informed about concern the <object type> he has specified, and any of the superclasses of this <object type>.

form2 When a DESIGN request is issued, only those agents interested in performing a certain design function <design-method> for the specified object type <object type> are informed.

Note a DESIGN may be requested for SELECT_OR_SYNTHESISE, and two INTERESTS may have been indicated by an agent for SELECT and one or SYNTHESISE.

RECEIVED

Syntax: *form1* RECEIVED <agent> PROPOSAL <design-method> <design-object>

form2 RECEIVED <agent> EVALUATION <proposal>

Parameters:

<design method> either PARAMETRIC, SELECTION, or SYNTHESIS

form1 An indication that a proposal was successfuly received from agent <agent> for the

<design-method> of <design object>.

form2 An indication that an evaluation proposal was received from agent <agent>.

UNABLE_TO_PROPOSE

Syntax: UNABLE_TO_PROPOSE <agent> <design method> for <desObj> [as [initial | alternative]]

When an agent <agent> has specified an interest in presenting a proposal, and cannot do so (may be because certain structural information is not available) then the agent responds with UNABLE_TO_PROPOSE the <design method> design of <desObj>.

'INITIAL' or 'ALTERNATIVE' is required so that we know that the agent has responded. If INITIAL we need to inform the WAIT objects, otherwise not. If an alternative is not available to be proposed, the NEGOTIATION object will be triggered to analyse the situation, and potentially request another agent to put forward a proposal.

UNABLE_TO_EVALUATE

Syntax: UNABLE_TO_EVALUATE <agent> <proposal>

When an <agent> has been requested to evaluate a proposal <proposal>, we will expect a response. In this case the agent can not evaluate a solution and therefore informs the negotiation mechanism that it is unable to evaluate a solution. When all proposals have been received, then the option proposal is informed that the review is complete (review-complete).

WAIT OBJECTS

An object is created when a DESIGN request is made. When requests to design are sent to the agents, AGENT_DESIGN, a record is made in the WAIT_ON_PROPOSE in the knowledge that it should respond. When an agent reponds with a proposal then the agent reference is removed from the WAIT_ON_PROPOSE object. When no more agents are recorded as requireing to propose, then we know that the design can procede.

OBJECT: WAIT
Operations: remove-agent
add-agent

any-agents-left

OBJECT: WAIT_ON_PROPOSE of type WAIT
Operations: design-method
design-object

<design method> is one of: PARAMETRIC, SELECTION, or SYNTHESIS

OBJECT: WAIT_ON_EVALUATE of type WAIT
Operations: design-proposal

Appendix M. Design tasks in the engineering design of process plant

Chemical Path Synthesis

Chemical plant synthesis is essentially a research and development technology function and is concerned with the invention of chemical reaction sequences leading from the available feedstocks to the desired product [Reklaitis, Preston 89].

Process design

Process design has traditionally been viewed as a three phase function [Reklaitis, Preston 89];

i. Selection of the sequence of chemical and physical steps or unit operations to be employed to realise the synthesis path

ii. Assignment of equipment types to unit operations

iii. Definition of flowsheet structure, operating conditions, and functional equipment sizing.

The process engineers are responsible for the design of a process and that it operates efficiently and safely under continually changing conditions.

The process engineers are usually the first people on the scene when a new contract is being discussed with the client. Their role at this stage is to weigh up the scale of the problem and make an estimate of the effort required. During most of the early stages in design the world evolves around process engineering, with all disciplines requiring initial design details in order to begin their work.

The output of the process engineering function consists mainly of the process flow diagram (PFD) complete with sets of heat and material balances and process specifications for all major items of equipment. This together with a variety of other documents is called the process package. It has been estimated that less than 10% of the total plant cost is devoted to this work and the decisions made at this stage account for over 80% of the total capital costs [Winter 92]. See Appendix A for more detailed coverage of this design task.

Safety reviews and heat integration

During the development of the PFD and P&ID there may be several reviews of safety and heat integration. These studies may occur at other stages throughout the project when deemed necessary.

Safety studies are termed *Hazard and Operability Studies* or *Hazop* studies for short. The Hazop technique was developed by ICI and is a structured and formal approach to identifying hazards with the design of a chemical plant. The method is essentially a brainstorming approach to identifying faults in individual lines throughout the plant that are depicted on the P&ID. The method is very time consuming and requires the involvement of individuals from each discipline.

Traditional methods of design used to separate parts of the process and design them individually. The industry have now realised that this design method is wasteful of energy as particular parts of the process are producing heat while other parts of the process require heat. *Heat Integration* studies are therefore designed to identify the re-use of energy throughout the process. Integration studies are most commonly performed from the PFD although these studies may continue in the development of the P&ID's.

The output of the Hazop or the Heat Integration study may result in changes to the P&ID.

Mechanical design

The mechanical engineering team is responsible for putting together the *procurement packages*. The procurement package is a set of detailed equipment data sheets that specify all the equipment necessary to build the plant.

The possible organisation of a mechanical engineering team will consist of a manager, a package engineer and a variety of discipline engineers [SfK 92d]. A package engineer is essentially a mini-project manager and is responsible for ensuring that all items of equipment are purchased. A discipline engineer is responsible for a specific group or type of equipment. A group of equipment may be 'rotating equipment' which would cover pumps, compressors, air coolers etc. A type of equipment may be vessels or heat exchangers that are of such a complexity that they require individual attention.

The equipment data sheets are specified by the relevant discipline and require information from the PDS, PFD and GA. Much of the information on the PDS is transcribed onto the equipment data sheet. The equipment data sheets make up the *enquiry package* that are distributed to the appropriate vendors. A vendor is selected from an approved list of vendors and from review of the vendors catalogues. The approved list of vendors is usually maintained by the contractor or the client.

After the vendors have put forward their proposals for supplying equipment the proposals go into the *bid evaluation* stage. At this stage each bid is reviewed from a technical and a financial viewpoint. The technical review is to ensure that the bid has met the specified process requirements. The financial viewpoint is considered by a another group responsible for purchasing the equipment and usually selects the cheapest.

During the development of the mechanical data sheets their is much input from other

disciplines, the materials, electrical and control group for example. Electrical engineering may sometimes specify part of the mechanical data sheet, the 'motor data sheets' for pumps (and various rotating equipment) for example.

The mechanical engineers are also involved from day one. They require information from the process group on the major equipment that is likely to be required to realise the process. There is much iteration at this stage as the process engineers do not have any precise details on what is required and the requirements may change. The vendors who supply the equipment are also aware of these problems and leave the construction of the detailed parts of the equipment to the last minute.

Materials selection

One group involved in the design and have not been mentioned are the *Metallurgists*. There role in the design is not straight forward and cannot be nicely incorporated into the life cycle.

The Metallurgists can essentially be involved at any stage during the design. Initially they may be involved as early as the preliminary costing stage for the plant. They identify the kind of materials that may be required to construct the plant. If the plant cannot be made largely out of carbon steel it is not likely to be cost effective.

A significant proportion of a Metallurgists time may be spent after the process flow diagram has been completed and the heat and material balance sheets are available. From the heat and material balance sheets he can view the temperature at which the process will be operating and see the various chemicals in each stream and the proportion.

The temperature can be an extremely important factor. A specific chemical may not be harmful to a material at one temperature but extremely corrosive with just a few degrees increase in temperature.

Metallurgy is a very skilled job. There are an infinite amount of chemicals and thousands of different materials. When selecting the material they have to consider the lifetime of a plant (which signifies the allowable limit on corrosion) and identify the cheapest material available to satisfy the requirement.

Piping design

The piping department is by far the biggest team involved in the design of a plant. It is estimated that about 50% of the total project hours are spent in piping design in comparison with the process group that account for 10%.

Piping is very much related to process engineering. The piping drafters must understand

the various process requirements better than the personnel in other areas of work. The main functions of piping are to [Rase, Barrow 57]:

i. layout, arrange and design all piping in accordance with the specifications and the applicable codes

ii. check drawings other than piping for clearances with structural steel. foundations and other types of equipment

iii. Study of the piping arrangements for stress

iv. list and specify all mechanical expansion joints

v. design, select and list all detailed pipe supports

The drafting process has benefited in recent years from the development of 2D and 3D CAD systems. The main output of CAD systems are the Isometric drawings (or isos) that are an accurate and most common representation of piping systems.

There are four basic types of isometric drawing [Lamit 81];

i. *System Isometrics* - indicates a complete or partial view of a plant.

ii. *Field Fabrication Isometrics* - indicates only the critical aspects of the system such as major items of equipment

iii. *Shop Fabrication Isometrics* - shows all face to face dimensions for valves, flanges, fittings including dimensions for placement and fabrication.

iv. *Detail Isometrics* - specific detail is highlighted (steam tracing, special trims etc). Detail isometrics are usually only drawn for a specific portion of a line.

Stress calculations for a line are also an important task to be performed by piping. The necessary supports and expansion joints need to be fitted into order to withstand varying loads and temperatures. There appears to very little optimisation of a pipeline [SfK 92d]. The attitude appears to be 'if the technology fits then use it'. Careful consideration has to be given when placing pipes between items of equipment because of the varying temperatures that may cause metals to expand. If a straight pipe is connected between a heat exchanger and a vessel for example then elaborate nozzles are required at both ends of the pipe to take the stress. In most cases it is not cost effective to have such nozzles and bends in the piping have to be considered.

Civil/Structural

The civil/structural team are responsible for the architecture of the plant, foundations and all industrial type buildings. These activities include designing the structural steel framework, sizes and connection of structural members, minimum clearances, handrails, ladders, stairs, floor plans, drains, fire mains, paving, trenches, utility buildings and so forth.

The civil structural team will work closely with piping, especially on the underground layout and fabrication. Underground supports for cranes also have to be designed so that the whole plant can be constructed.

Loss Control

Loss control is primarily concerned with the safety of the plant, sometimes referred to as the Safety and Loss Prevention group (SLP).

There role covers many areas of safety, a few of which are covered in the following list:

i. design of sprinkler systems and gas alarms
ii. access and escape routes
iii. fire and gas studies
iv. fire protection studies
v. area protection layouts

Safety factors should permeate through all stages of design and all engineers should be made aware of their responsibility. Loss control usually take part in directing hazard and operability studies and ensuring that safe design procedures are followed by the various disciplines.

Instrumentation

Instrumentation, as its name implies, deals with the instrumentation selection, fitting (ex. wiring plans) and control system design. The duties of the group can be summarised as follows;

i. production of general arrangement drawings for instrument installation and the various conduit runs

ii. listing of all instruments required in the plant together with appropriate codes, location, type of instrument, connecting equipment (air, electrical etc).

iii. preparation of drawings for the installation of instruments

iv. telecommunications facilities

v. central control system design and layout

vi. preparation of schedules for instrument installation

installation of fire and gas detection systems and the production of the cause and effect matrix

The group requires a diverse set of skills from process and electrical knowledge through to operating knowledge. The team are also likely to be involved in safety analysis studies to ensure failsafe systems are specified and operating procedures are documented.

Construction and fabrication

When all the design drawings and specifications have been completed the job of building the plant is then assigned to a Construction Superintendent[Rase, Barrow 57]. The job of the project engineer is essentially over.

The construction superintendent is likely to become involved in the project as early as preliminary engineering data is available. This will enable him to identify the major construction equipment that is needed to build the plant and determine the most practical construction methods.

The major stages of construction may proceed as follows [Rase, Barrow 57];

i. site preparation
ii. erection of temporary buildings
iii. excavation
iv. installation of underground facilities
v. foundation construction
vi. erection of major equipment
vii. installation of piping
viii. cleanup

The project engineer will however require to know information concerning the construction of a plant so that he can foresee construction problems earlier in the design stage.

Appendix N. Transcript of the vessel design meeting

The following is a transcript of the discussions in the vessel design meeting between two engineers Jeremy and Roger. Roger has a background in process engineering and in the case scenario is playing the role of a process engineer. Jeremy has a background in design, and is an expert on safety issues regarding design. Jeremy has considerable knowledge regarding current legislation and safety standards. The names of the engineers have been changed in this design case to maintain anonymity.

We've got a PFD and a toluene storage tank facility. We're going to attempt to develop the ELD from this. We took the level measurement device of the PFD because in most cases this would'nt appear on the PFD.

The first question that Roger would ask is, "why do we need a storage tank ?" (question the reasoning behind the PFD) - everyone is cost conscious at this time - may be it can just be tankered in. We're not connected up to a pipeline and therefore no reliable continuous supply so we need some means to continue production if deliveries can't be made for example. For this example we assumed that we needed enough inventory to keep the plant running for two days. How reliable is the just in time delivery ? As it's best to reduce stocks. Other factors to consider are mass storage where a product can be purchased and stored at a time of year when it is cheap to purchase.

Roger then wants to know how much to store. This gives him an idea on the size of the facility. Then he want's to know WHERE. In ICI the location is likely to have been sussed before the process engineer is involved and therefore the process engineer probably won't consider the location. Roger and Jeremy then think that SHE considerations should then be considered. They cover a large area (COSH, SIMA etc). These guidelines will determine if you can store a product, and how much. The ICI hazop one considers most of these considerations - although of course most people don't do this. One problem in large organisations is that engineers assume that others' have reviewed particular problems - when they may not have.

When considering location there are a whole host of considerations. Is the ground contaminated, what is the effect of leakage on the local population, is the ground stable etc. There are a host of hazard conditions that can be considered. Roger argued that at this stage the kind of hazards considered are the PRE-PFD, to distinguish between the types of hazard that lead to specific design detail such as: given that overpressure can occur lets have a pressure rupture disk on the tank. The PRE-PFD types of hazard are the "can we go ahead with the design as it stands" kind of evaluation.

From the example, Roger and Jeremy were concerned with ground contamination. There are really two considerations. Firstly, what if you pollute the land with toluene ?, and secondly, is the ground already contaminated and what would happen if toluene were to mix with this chemical already in the ground ? (I assumed from the comments that it is not

worth cleaning up).

ICI had an earthquake in Cheshire a couple of years ago which caused problems. This is a constraint that you should consider for the site - and not really for the piece of equipment. This requirement however should have been considered at HAZOP study 1. If the tank however may be particularly sensitive - i.e. need securing - the process engineer may consider asking the question concerning earthquakes.

The engineer should consider regulations, company standards, site standards etc. The process engineer would have to ask questions to the particular customer to ensure that he knows what standards to apply.

Jeremy thought that a young engineer would get board specifying this vessel and would just want to get down to how to store and provide 200 tonnes of toluene. This however is the wrong attitude as Roger mentioned that this may be one of the most critical elements on the plant.

[Jeremy] The process engineer then asks is it dry?, is it pure toluene?, what is the temperature ? The process engineer is asking the questions in order to select the tank. For example, if the toluene was coming back hot, and had a vapour pressure of 3 bar - you can not store it in an atmospheric pressure tank. Roger's comments were that the aim was to collect all the information in order to design and sketch out the tank. You would want to know the flashpoint - in order to determine type of flame traps, nitrogen blanketing etc. Jeremy then identified that with mecro-ethalulate - even though it is flammable (which is why you would nitrogen blanket - so you wouldn't have a flammable atmosphere) needs contact with air otherwise it will polymerise. Jeremy then considered where the nitrogen would go to - may be to a thermal oxidiser - although this presents a back pressure problem. It may well be cheaper instead of putting fans in and all the problems with that, to design it to be a **pressure vessel** and blow vapour through to an incinerator. One of the problems that we are coming up against is that with an existing tank that is designed up to 18inch water gauge you can't blow the gas directly through to the incinerators so you would have to put a suction system in which can draw air in which would create a flammable atmosphere, create flammable headers and call kinds of complications, whereas if we know that it has to go to a thermal oxidiser we would be designing a tank that could vent easily to thermal oxidiser. So we have to decide where we are going to vent to ?

All these considerations reflect on the design pressure, which reflects back on the type of tank. If we have nitrogen blanketing we would still have to vent back to an incinerator. If we thought that in the long term we would never connect this to an incinerator we may settle for an atmospheric tank, but then vent to a flare. The normal design pressure for an atmospheric tank is 8 to 10 inch water gauge. If were looking at an atmospheric water tank and someone says that the design pressure is 12 inches water gauge, then this is the *maximum* pressure, so its operating pressure would probably be around 8 inches water gauge. So if you have an emergency vent manhole on it, the pressure in the tank will go

up to twelve in a fire and your vent will be fully open. Collapsing the tank is simple - this can be done by putting a polythene bag on the vent, or for example piping in cold water into a tank full of steam.

From the shape of the tank on the process flow diagram, it is drawn as an atmospheric tank. Roger and Jeremy are questioning whether this is appropriate - because toluene is unlikely to be allowed to vent to atmosphere.

You have to consider the type of tank because for instance, overflows can only work on atmospheric storage tanks. For high pressure tanks you would have high pressure alarms or reliefs etc - therefore requiring a different design. What Roger and Jeremy are trying to do is ask all the questions necessary to determine whether an atmospheric storage tank, or high pressure vessel is more appropriate.

Jeremy identified a further problem because we are designing for pressure. This consideration is whether the vents can block - this is important in selecting the right tank. If the fluid can crystallise, and to design it to be inherently safe you have to design the tank to take the dead end pressure of what can be feed to it. We can't deal with fire relief but we can deal with other conditions pertinent to it. At this stage we have to ask the question of whether blockage of the vent is likely.

The consideration of heating coils is important - especially in winter conditions for example. You have to know whether there are traces of substances, take wax for example, - the low melting point material. The chances are that with toluene you wouldn't put it in as the freezing point is quite low isn't it ?. It's an important question to ask, for example in china you can get down to temperatures as low as -30. In the UK what is the minimum temperature likely to be ? There is published information around but where do you get it from ? You could ask the standards department but they may not know. Atmospheric information is around, and this data would identify whether we would need any heating coil to maintain a minimum temperature in the vessel.

[Jeremy] An interesting problem with this case is that there is a recycle loop back into the inlet vessel, we have to be careful not to have a pressure vessel that is not designed to cope with this. Are there any other high pressure sources into this tank ?

It is clear the Jeremy and Roger are attempting to consider many things at once here - the possibility of back pressure, whether the tank should be nitrogen blanketed, and where the tank should be vented to. This impacts on the type of tank - atmospheric or pressure vessel. The PFD shows an atmospheric tank in the example, and they are both questioning whether this is the right approach. Another question pertinent to the type of tank is whether vents will block - this effects the design pressure rating.

[Jeremy] we could have floating roof tanks. If you have a floating roof tank you wouldn't have to worry about venting. It is appropriate in situations of low volatility and low

toxicity.

[Roger] In a real life engineering situation you would ask all these questions, and the chances of getting answers back are small. When Roger is requested to design a tank, he would right all these questions down. However, when you get into the nitty gritty design detail, you find you have missed questions out and keep having to go back for more and more information. Some information is readily available.

You have to question having a storage tank. There are distinct disadvantages of having a storage tank on site with large quantities, as well as the cost. Because we have a requirement for 10 tonnes an hour, then you can't meet this without storage. There is going to be a lot of tankers, and a lot of road traffic coming in and out.

On the PFD you only have to have one storage tank is shown. On the ELD, you may show more than one. If you have to take it down every two years to clean out the sludge, you have to have two tanks minimum. There are other considerations that come into that for example, how much does it cost ? The number of tanks really comes down to methods of operation as well. Quantity itself can be broken down into any number of tanks. For example, do you want to fill one tank up and run one down ? - or have the facility to have one tank out for inspection or maintenance, or if you have a problem with one tank because its leaking, you have another one to draw from. You could just build one great big tank and leave it at that. You may have two tanks with one left empty so that if you have a leak you can leak into the empty tank - this has happened in some cases. The location has also to be considered when deciding number of tanks. Roger was unsure as to whether regulations stipulate maximum storage quantities. CIMA regulations stipulate maximum quantities allowed before you have to get permission from the authorities.

Another problem is cross contamination - say one person loads it up with another fluid which is not toluene. This goes down to more detailed questions such as tanker connections and is it possible to load it up with another material and what are the consequences. Given different connections for the fluid - loading up with the wrong product would have to come down to *sabotage*.

A lot at the end of the day comes down to cost. Many items of equipment are standard. If you specified all the specific design detail, say for example specifying the number of seats, type of windscreen wipers etc, when buying a car - it would cost you a fortune. There are standard tanks to choose from out in the market - even though they are mainly tin-cans.

(Both Roger and Jeremy summarised some of the above discussions and clarified their understanding.)

Do we need to do it in the first place ? With 10 tonnes an hour, we need road tankers queuing up, connections to make, which is all quite hazardous and you would not want to

do that. We need a nice large tank - keep it topped up so that people don't panic about connecting the tanker pipes within 5 minutes for example.

How much material do we want to store ? 500 m3 - basically two days supply. Do we comply with SHE for storage. You may purchase the stock at a cheap time of the year and pay someone to store it off-site - so that you don't have it near your process facility.

Location ? Existing petrochemical works near plant.

Planning ? planning permission will be needed as we are building the plant. SHE/COSH implications. Material data sheets will have to be available. CIMA not considered as it is really for large installations (say a 5 tonne sphere of chlorine).

Hazards ? Flammable, flashpoint 10 degC?, toxic, residue ? Jeremy identified a problem that we had not considered contamination and there is benzene in the process - this effects the flammability. The contamination may be a function of re-cycle - rather than actual feed.

Nitrogen blanketing - will we need it ? Yes - because it is flammable. Flashpoint is 10, below maximum operating temperature.

Where do we vent to ? [Jeremy] a thermal oxidiser, the example does not show a thermal oxidiser, however this is 1994 and we're not allowed to vent toluene to atmosphere. [Roger] are we sure though ? [Jeremy] yes, and we need to decide this now as it is part of the original approval - what are we going to do with our vents. [Roger] Won't it depend on the quantities though ? [Jeremy] I don't think we would get approval nowadays. [Roger] so we would have to have a pressure vessel then ? [Jeremy] well it doesn't stop us having vents. [Roger] well it would, the only way it would stop us if we had a floating roof vessel. [Jeremy] but we can't have a floating roof vessel because of the high volatility and toxicity.

[Roger] so we need a vent because it is an atmospheric tank, and because the tank has to breath. We need a thermal oxidiser because of regulatory requirements.

[Roger] (Type of tank and design pressure). Well we've said its an atmospheric operation. I'd be reluctant to say we should put a pressure vessel in their because of cost, its a big tank & costs a lot of money. It comes down to a cost question, how much pressure do we need to get it to the thermal oxidiser ? [Jeremy] or we could have a suction system, however we now have the possibility of the suction system sucking your tank in. [Roger] yes, an atmospheric tank won't withstand suction. [Jeremy] there are back-breakers, that blow when the tank is under suction. Problems occur with a system under suction, and when you have a flammable vapour coming through which can be diluted with air to make a flammable mixture, then you have something going towards the thermal oxidiser which is flammable mixture, which can cause detonations and explosions in the vent header. You

also have to allow for when the thermal oxidiser isn't working, or for some reason can't take stuff. You then have to have vent valves to vent to atmosphere or a flair stack, and if that vent valve is open when it shouldn't be open, then you can suck in air through that and therefore have a flammable atmosphere. [Jeremy] the decision of saying that it is not a pressure vessel is interesting. [Roger] this is what I thought I would do ! Whether we have to vent to a thermal oxidiser is a fundamental question, determines what the tank is going to look like - whether we are talking about an atmospheric tank or a pressure vessel.

[Jeremy] if you have the tank as a pressure vessel, and the vessel had more than a couple of bars in then you won't get the round tankers to discharge. [Roger] if I was designing it. if we stick with assuming that we want to go to a thermal oxidiser, I would be wanting to know what sort of pressure would I need the vapour to go to the oxidiser at. I would put down a list of considerations like *pressure to oxidiser*, lets say .2 bar. [Jeremy] What about method of filling - lets assume blown tankers (pressure on the top and blow it out) at say 2 bar gauge. [Roger] Is this how you would normally empty them, I would of thought pumping was appropriate. The options are to either pump it or use nitrogen pressure. [Jeremy] well it would be air from a tanker because they haven't got nitrogen on the tanker have they. [Roger] Well a case I have worked on before, the nitrogen was actually provided by the works. [Jeremy] you then have the problem of when its empty, you have all this pressure with an open end through the pipe, up into the storage and out through the vent. [Roger] well the way to get over that is through a trip valve. [Jeremy] well that doesn't always work, it may jam open, you have to design to cope with it. Filling from road tanker is therefore a pressure consideration that has to be accounted for. We therefore have pressure to oxidiser, pressure from road tanker, and pressure from other sources, in this case re-cycle to consider. [Jeremy] the liquid will normally effect the design pressure - [Roger] but lets have it as a consideration.

[Roger] the recycle to the storage tank from the plant is interesting, I have never seen that done before. [Jeremy] crackers are a different case, you never crack everything, you always feed back. The feedtank in this case can be contaminated with a lot of rubbish. [Roger] In most cases what comes out the storage tank should be pure.

[Roger] is blockage of the vent breather likely ? [Jeremy] no, not with toluene. Material properties will tell you this. With some materials you have to design more maximum pressure as you can't be sure of keeping the vents free. [Roger] well there is no polymerisation - but another problem I've come across is birds nesting in vents. [Jeremy] its not a big problem, you can design against this.

A thermal oxidiser is just a flame and you put the organics into the flame. The oxidiser works under controlled conditions with guaranteed residence times. We can guarantee to kill all the halogens etc at a certain temperature for a certain length of time. With a flare stack all you are trying to do is get a reasonable amount of combustion.

[Roger] we've assumed the liquid is pumped from the tanker, therefore we won't get gas

breakthrough from the road tanker, but because we have a re-cycle from *still* we need to know whether gas breakthrough to the tank is possible. [Jeremy] if we did loose level from *still* and blow through, we would end up with 6-8 m cubed an hour of vapour from the still (distillation column). [Jeremy] what we are asking is would you design the tank on the basis of the still going wrong or would you design the relief on the basis of that going wrong.

[Roger] this example is a good case - as you never get all the information that you need (with ref to venting VOC's). An engineer will develop a design and at a later date someone will notice that you have vented to atmosphere and tell you that you can't do that any more. This information may not have got through to the process engineering guys. So you then go away and investigate and if someone says that its not allowed, and bearing in mind that at this stage you are probably at HAZOP, you then go back and work out what to do now. At this stage things are being built, concrete poured and things are more or less there. Well the vent, we are going to get suction so what are we going to do ? We are going to have to put a fan in or a blower to get the vent stuff back to the thermal oxidiser. We've got a thermal oxidiser now that we thought we didn't need, so instantly we've two more pieces of kit. We've now got all the other implications that say, because we have got a blower on and we are purging the thing with nitrogen then we have to beef up the nitrogen flow to make sure you don't suck the tank in. The next question is what if the nitrogen flow fails. You then stand a good chance of sucking the tank in. To overcome this problem, we then have to ask if the nitrogen fails what is going to happen ? Do we have some system on that trips the blower to the thermal oxidiser because that's another piece you have to put on, or if we can get away with air being sucked in for these sort of abnormal conditions, and we're happy to live with that - you need to put a back breather on. So as a minimum you have your thermal oxidiser, your blower, and you've got some system on to ensure your nitrogen flow or a back breather - so you've instantly got another three pieces of kit - and a lot of heartache and head scratching about the design of it. Now if we had addressed this problem earlier, we could have probably got away without a blower, a back breather and some sort of trip system - but it might mean increasing the design pressure of the tank. You might even still say you will go with that, because cost wise it is a lot cheaper than building the tank as a pressure vessel.

[Roger] Minimum temperature - do we need heating coils. We said the minimum temperature is 0 degC. We therefore don't need heating coils and because of hot re-cycle back from plant.

[Jeremy] An interesting thing on the PFD is this little thing on the bottom of the tank. It looks as though it is trying to trap solids. I'm wondering if there is something about the material that we have not asked about. We have to find out about solids or possible decomposition. This is very important to how we design a tank - as these solids will have to be removed from the tank and they may be toxic and carsneogenic. We may want to have a specially designed tank to enable easy removal of these solids. You may want to heat the solids to remove the vapour.

[Jeremy] lets assume two tanks because it will need cleaning out on a regular basis. Other reasons will cover availability and inspection. [Roger] I would assume that they would remove the residue by just opening a valve at the bottom of the tank without emptying the tank. [Jeremy] it depends on the type of residue. [Jeremy] lets say that it builds up and we have to wash it out. We have to know all of this to develop the ELD to identify special utilities such as heating coils. Also if you have heating coils, you have to design the tank to work at 100 degC for example. Everything impacts on everything else.

[Roger] Method of operation, or operating philosophy. Do we want one full and the other empty, do we want them both full. Lets say normally balanced for less chance of overfill. So for cleaning you can empty into other tank. [Jeremy] so where is our re-cycle going to ? [Roger] we would have to have a re-cycle to each tank wouldn't we ? [Jeremy] its a good question - if it does not split itself properly we could have problems. [Roger] we need some method of flow control. [Jeremy] ohh - thats money! - another 10 thousand pounds. We have to ask some more questions to determine whether a control valve is needed. [Roger] we might be able to design it so that we don't need a control valve - but I'm not quite sure how. [Jeremy] does it matter if it all goes back into one tank - because the composition would be completely different in the two tanks (but neither Roger or Jeremy knows). You would have to look up safety data sheets, and vapour pressure.

[Roger] How do we cope with emergency conditions. There's fire, loss of nitrogen, loss of thermal oxidiser, and reverse flow from benzene column, loss of nitrogen pressure control ? (we left these as considerations).

[Roger] what about materials of construction ? [Jeremy] that includes joints and everything else - what is allowed and what is forbidden. Sometimes you have a substance where you must not have aluminium on a plant - even if the metal does not come into contact with the fluid. We also need to consider the special handling requirements to do with the materials of construction (the velocity of sulphuric acid for example can strip of the passive protection of a pipe). [Roger] we're ok for building it in mild steel. I'll put a system on here (when drawing the ELD) to allow for in-breathing and out-breathing when loading from storage. Would we put an isolation valve in there ? [Jeremy] I assume we will for maintenance of the tank. [Roger] the thing that worries me their is that someone could go over and shut it. [Jeremy] yes, but the pressure control would still pick it up you have an indication if there is a problem. [Roger] would it stop sucking the thing in ? [Jeremy] we have the valve for basic requirements - if you want to add any alarms or anything ..[Roger] why don't we put a back breather on ? [Jeremy] well you may have to. these are the kind of problems you may have.

[Roger] I think Jeremy is thinking about it from an operational point of view. (some irrelevant discussions.....)

[Roger] We know that we have got tanks coming in, and we know that we are going to off-load with a pump. When it comes to thinking about where you are going to put things

like valves, obviously when you come to pump this stuff in, you want to be able to disconnect your system safely, so just say for example, you had a connector or some sealed couplings, when you have taken that off you don't suddenly want to get reverse flow of liquid and all going over the floor. There are also things about the tanks themselves, just forgetting about valves, when we put liquid in you don't want splash filling. You would be surprised that although this knowledge is standard stuff, many people forget about it. [Jeremy] you could put in a dip pipe with an anti-syphon hole. [Roger] what I was thinking about is that you could avoid splash filling by feeding in at the bottom. [Jeremy] but then you get back flow. [Roger] well we could get back flow from a syphon. [Jeremy] depends on where your syphon hole is. [Roger] so if you put a dip pipe in you have to have an anti-syphon hole ? [Jeremy] yes. Well, you don't have to. [Roger] but it's better to have it isn't it ? If we are thinking about designing plants safely - which is what we are about.

[Roger] right, considering splash filling and static buildup. If we were storing water it would not worry us. [Jeremy] we might want to ask questions about frothing. Presumably if you do splash filling you may get frothing - a nice foam on top of it. [Roger] any other problems with splash filling ? [Jeremy] Another problem is earthing, and this is relevant to the whole installation as to whether you have to do earth bonding. You have to put the bond across the joints and sometimes if you are using a non-metallic system earthing electrodes or a ring to place in to earth the fluid itself. There are systems that are so corrosive that there are no metallic parts in it, so you have to put in a very exotic ring to earth the fluid.

[Roger] so we have the choice of bottom filling, dip pipe and anti syphon. [Jeremy] with bottom filling there is a risk of dumping 250 tonnes of stuff down to your filling point.

[Jeremy] may be we could put a refrigeration condenser above the tank to reduce the emissions. [Roger] thats a better idea I'll go for that. [Jeremy] you may well do, if you have not got any other major vents to an incinerator, the thermal oxidiser is a very expensive option and institutes a lot of problems - refrigeration may be a possibility. [Roger] why did we have thermal oxidiser ? [Jeremy] SHE is what it is all about at this stage, before you get to far down the track into all the details. [Roger] the question is not that we might not be allowed to put any out, but how much VOC we are allowed to put out. This is an interesting question if the tank already exists, if your venting VOC's what options are there around to reduce this. A lot comes back to the regulations, what are we advised to do, and what do we have to do. Time also comes into it. If in the future tighter VOC rules are applied, the thermal oxidiser will always be an acceptable solution, whereas refrigeration condensers may not be.

[Jeremy] How do we ever do design ? [Roger] It's like the toad and the centipede. The toad said to the centipede which leg do you put forward first, and he never walked again. How do we design a storage tank.....I don't know we just did it!